**中图分类号：TP319 论文编号：**



**工程硕士学位论文**

**基于SOA架构的客户精扫系统**

**研究与开发**

**学 号**： SF1316004

**姓 名**： 王翔宇

**专业、领域**： 计算机技术

**研究方向**： 计算机软件技术­­­

**指导教师**： 蒋夏军 讲师

**南京航空航天大学**

研究生院 计算机科学与技术学院

二○一六 年 九 月

Nanjing University of Aeronautics and Astronautics

The Graduate School

College of Computer Science and Technology

**Research and Development on Customer fine sweep System Based on SOA architecture**

**A Thesis in Computer Technology**

**By Wang Xiangyu**

**Advised by Jiang Xiajun**

**Submitted in Partial Fulfillment of the Requirements**

**for the Degree of Master of Engineering**

**Sept, 2016**

**承 诺 书**

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

本人授权南京航空航天大学可以有权保留送交论文的复印件，允许论文被查阅和借阅,可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或其他复制手段保存论文。

(保密的学位论文在解密后适用本承诺书)

作者签名：

日 期：

摘 要

当前，国内电信市场发展迅速，运营商需要建立更加精准高效的客户资源，以便适应市场需求。客户精扫系统是运营商为了适应日趋激烈的市场竞争和变化极快的实际需求而产生的新型客户资源信息管理模式。面向服务架构(SOA)是当前运营商处理复杂业务、实现企业资源有机集成的最有效的软件架构解决方案。本文在深入研究面向服务架构技术的基础上，通过探究客户需求、面向客户拓展业务、满足消费群体从单一化转向多样、业务处理从大众化转向成精准化、个性化和智能化的客户资源管理新要求，提出了一种快速构建客户精扫系统的解决方案。

本文对分布式SOA开源服务框架技术进行了深入的研究。在全面分析系统需求的基础上，提出了利用Dubbo服务治理框架，全面整合MQ异步队列消息框架和分布式缓存等技术开发客户精扫系统的新方法。系统后端采用Dubbo分布式SOA框架和Zookeeper注册中心构建完整的分布式同步调用框架，异步消息框架采用RabbitMQ构建。前端主要采用HTML5技术，Web界面采用jQuery和Bootstrap构建。

在前后端之间通过多次迭代和调整，加入服务端前置中间层，有效地实现了系统层面解耦，降低了前后端通信的复杂性。在后端Dubbo分布式框架中整合Zookeeper注册中心很好地解决了服务的订阅和发布、服务的动态发现以及透明化路由。

系统运行测试表明，性能稳定、响应快速高效，能够应对高并发、大流量的客户访问；能够适应业务功能的快速迭代和横向扩展需求，为后期系统的升级维护提供极大的便利；能够有效集成运营商企业系统的各类资源，构建稳定高效、易维护、易扩展的企业级应用。

**关键词**：客户精扫系统,运营商集成应用,SOA服务治理,Dubbo框架,Zookeeper服务,分层架构

**ABSTRACT**

At present, the domestic telecom market is developing rapidly, and operators need to establish more accurate and efficient customer resources in order to adapt to market demand. The customer fine sweep system is the new customer resource information management pattern which the operator produces in order to adapt to the increasingly fierce market competition and the fast changing actual demand. Service-oriented architecture (SOA) is the most effective software architecture solution for operators to deal with complex business and realize the organic integration of enterprise resources. Based on the deep research of service-oriented architecture technology, the paper explores customer demand, expands business to meet customer's needs, transforms business process from popularization to precise, individualized and intelligent customer resource management requirements, and proposes a rapid solution to build customer intensive system.

In this paper, distributed SOA open source service framework technology is deeply studied. On the basis of comprehensive analysis of system requirements, a new method is proposed to exploit the Dubbo service management framework, and to fully integrate MQ asynchronous queue message framework and distributed cache technology. Dubbo Distributed SOA Framework and Zookeeper Registry Center are used to build a complete distributed synchronous invocation framework. The asynchronous message frame is constructed by RabbitMQ. Front-end mainly adopts HTML5 technology, and Web interface uses jQuery and Bootstrap to build.

The middle layer of the service-side is added between the front and back ends through many iterations and adjustments, effectively achieving the system level decoupling, and reducing the front-end communication complexity. The back-end Dubbo distributed framework integrates Zookeeper registration center to solve a good service subscription and publishing, dynamic discovery services and transparent routing.

The system’s test which runs the whole process shows that the performance is stable, the response is fast and efficient, and it can cope with the high concurrency and large flow of customer access. It can adapt to the fast iterations and lateral expansion requirements of business functions. It can greatly facilitate the upgrade and maintenance of the later system. It can effectively integrate all kinds of resources of the operators' enterprise systems and build enterprise applications that are stable, efficient, and easy to maintain and expand.

**Keywords:** Customer accurate marketing system, SOA service governance, Operators integrated applications, Dubbo framework, Zookeeper service, Layered archtecture

目 录

[摘 要 I](#_Toc465275430)

[**ABSTRACT** II](#_Toc465275431)

[第一章 分布式精扫系统研究背景 1](#_Toc465275432)

[1.1电信市场营销背景 1](#_Toc465275433)

[1.2 客户精扫系统的国内外研究和开发现状 1](#_Toc465275434)

[1.3 构建分布式精扫系统的实际意义 2](#_Toc465275435)

[1.4 课题背景 2](#_Toc465275436)

[1.5 文章组织结构 2](#_Toc465275437)

[第二章 分布式架构技术基础 4](#_Toc465275438)

[2.1分布式架构技术起源与定义 4](#_Toc465275439)

[2.2分布式架构特性与技术 6](#_Toc465275440)

[2.3分布式框架主要实现方式 6](#_Toc465275441)

[2.4 采用SOA的服务化分布式架构实现系统构建 12](#_Toc465275442)

[2.5 本章小结 15](#_Toc465275443)

[第三章 精扫系统需求分析与功能模块设计 16](#_Toc465275444)

[3.1 精扫系统功能定位 16](#_Toc465275445)

[3.2 分布式框架在精扫系统中的应用 16](#_Toc465275446)

[3.2.1主动应对高并发 16](#_Toc465275447)

[3.2.2 可伸缩与负载均衡 17](#_Toc465275448)

[3.2.3 资源共享 17](#_Toc465275449)

[3.3 精扫系统功能模块设计 17](#_Toc465275450)

[3.4 本章小结 19](#_Toc465275451)

[第四章 精扫系统架构设计 20](#_Toc465275452)

[4.1 精扫系统结构与数据模型 20](#_Toc465275453)

[4.1.1 精扫系统分层模型 21](#_Toc465275454)

[4.1.2 精扫系统软件模块 22](#_Toc465275455)

[4.2 系统数据模型设计 23](#_Toc465275456)

[4.3 本章小结 24](#_Toc465275457)

[第五章 精扫系统功能设计与实现 26](#_Toc465275458)

[5.1 业务功能设计 26](#_Toc465275459)

[5.1.1 前端模块 26](#_Toc465275460)

[5.1.2 系统总控层 26](#_Toc465275461)

[5.1.3 客户服务模块 28](#_Toc465275462)

[5.1.4 产品服务模块 30](#_Toc465275463)

[5.1.5 订单管理服务模块 30](#_Toc465275464)

[5.1.6 市场营销模块 32](#_Toc465275465)

[5.1.7 报表统计模块 33](#_Toc465275466)

[5.2项目接口设计规范 34](#_Toc465275467)

[5.3数据实体类设计 37](#_Toc465275468)

[5.3.1 客户域数据模型 37](#_Toc465275469)

[5.3.2 产品域数据模型 38](#_Toc465275470)

[5.3.3 订单域数据模型 39](#_Toc465275471)

[5.4 基于Dubbo以及MQ的分布式服务调用实现 40](#_Toc465275472)

[5.4.1实际项目工程文件结构 40](#_Toc465275473)

[5.4.2项目Dubbo实现 42](#_Toc465275474)

[5.5设计基于Zookeeper的服务注册中心 46](#_Toc465275475)

[5.6项目RabbitMQ实现 48](#_Toc465275476)

[5.7 分布式缓存调用Redis调用实例 49](#_Toc465275477)

[5.8服务端前置层Front-Platform项目结构和设计方案 51](#_Toc465275478)

[5.9 本章小结 53](#_Toc465275479)

[第六章 系统测试 55](#_Toc465275480)

[6.1 系统测试部署 55](#_Toc465275481)

[6.2 RabbitMQ测试 56](#_Toc465275482)

[6.3 Redis主从集群系统性能测试 57](#_Toc465275483)

[6.4 本章小结 59](#_Toc465275484)

[第七章 回顾与展望 61](#_Toc465275485)

[7.1 论文工作回顾 61](#_Toc465275486)

[7.2 展望 61](#_Toc465275487)

[致 谢 62](#_Toc465275488)

[参 考 文 献 63](#_Toc465275489)

# 第一章 分布式精扫系统研究背景

1.1电信市场营销背景

电信运营商面对的客户具有全行业属性和覆盖广的特点。电信市场是国内具有巨大发展潜力与规模的市场，进入互联网时代，电信市场的营销思维方面需要进行灵活地转变，使其能够真正地适应时代背景和客户的需求[1]。

在目前竞争日趋激烈的运营商市场背景下，营销思维有着如下的转变：首先，要从“适应消费者的需求”转变为“根据企业、个人用户的特点引导消费需求”。其次，要充分利用互联网（包括移动互联网、云计算等等），利用信息化市场的先进手段和设备，从以往信息不对称的状态下走出来，在掌握海量营销信息的前提下展开营销。第三，过去的电信企业习惯于为大群体指定客户计划，在营销战略中缺乏针对性，而如今，随着个性化服务需求的提高，我们越来越需要真正的个性定制服务，将服务群体细化为不同层次和类别的客户群，针对每个群体对电信服务的不同要求创制不同的业务套餐，最大限度的满足客户的需要，也能提升市场对企业的认可度和企业自身的业绩，从而创造更多价值。

目前国内电信运营商客户市场环境的演变和市场竞争的加剧，对电信运营商组织客户营销，尤其是细分市场类别、精细化营销、个性化营销提出了新的要求，本文旨在通过分布式精扫系统来组织对客户市场细分化营销，搭建一个全面的分布式客户针对营销系统，基于移动设备（微信端、APP端）通过分布式服务的调用、实时地、移动化、全天候对企业以及个人客户进行精细营销。基于以上背景条件分析得出分布式精扫系统的理论需求与实际意义[2]。

1.2 客户精扫系统的国内外研究和开发现状

客户精扫系统是从客户关系管理（CRM）领域发展而来的。国外从上世纪90年代开始关注客户资源关系研究，Merlin Stone在《Successful Customer Relationship》一书中首次使用了“Customer Relationship Management”这一概念，但其研究重点在于如何在客户营销中的使用信息技术，对客户精扫概念未作明确阐释。学术界一般认为，客户关系管理（CRM）的理念是由Gartner Group公司于1997年首次明确提出，由此人们在CRM能有效提升企业管理水平、提升客户管理质量方面达成共识[3]。

本世纪初, 国内外很多公司开始在如何建立分布式客户精扫系统技术方面开展研究，以便更加精准的建立客户关系，客户精扫的理念由此而生。在客户关系管理初期，企业一般是广泛搜集和分析所有客户与企业的联系信息，大量精力消耗在与客户的接触方面，之后提出了有针对性地对客户资料进行详细分析，并设立客户精扫系统，提高客户资源管理质量。

总体来看, 客户精扫概念尚处于发展时期，主要用于更加精准高效地进行客户资源管理，客户精扫是一项企业管理职能，利用客户精扫系统，企业能高效地向目标客户推介产品，从而节约营销成本，提高企业效益。

1.3 构建分布式精扫系统的实际意义

传统意义上的电信系统，特别是客户关系管理、客户维系系统等等，一般都是基于传统的MVC（Model View Controller）三层架构进行搭建，但是并非物理意义上的分层，即使将应用架构分成三层进行开发，团队在不同层进行了代码实现，最终项目还是运行在同一台机器的同一个进程中，依然属于单块架构应用[4]。一般传统系统扩展方案是增加机器数量，搭建集群，利用负载均衡的方式来最终访问服务。

本文从技术原理上对“基于SOA架构的客户精扫系统”进行了相关研究，探讨通过SOA服务治理思路，针对传统垂直构建的企业应用进行分布式处理的理论方法，讨论在构建过程中引入分布式缓存、分布式服务框架等相关技术以实现新的客户精细化营销系统的可行性方案。为深入使用具有分布式特征的、移动互联网化的企业级应用、更好地为市场和营销服务做理论准备。

将传统完整的单块业务系统拆分为小的服务，将多次调用且重复开发的功能独立出来作为通用模块，将复杂的系统解耦、拆分为细粒度的服务进行独立部署，增强系统的重用性，有效降低应用的压力[5]。形成完善的分布式系统后，各个子系统之间相对独立，并且能够按照统一的方式对外提供服务，这样每个子系统互相之间不用关心具体实现，能够大大降低整体应用的开发难度，提高可重用性，缩短开发周期保证应用的灵活迭代升级，快速响应业务需求。最终使得IT系统能够不断快速地响应市场需求，提供精细化营销指导，高效满足客户需求，具有显著的实际意义。

1.4 课题背景

中国电信江苏公司自2007年使用CRM（Customer Relationship Management 客户关系管理）系统以来，客户产品关系与客户信息得以系统化管理。但随着传统意义上的营业厅模式逐渐减弱，移动互联网和精确客户营销模式的兴起，亟需开发面向精确营销的客户精扫系统。

本课题得到以下项目的支持：

1、中国电信江苏公司客户精确营销系统项目。

2、中国电信集团天翼电子商务有限公司，营销与代金券及个人帐户项目。

## 1.5 文章组织结构

文章在基于中国电信客户精扫系统项目实际背景下，对基于SOA架构进行了理论分析、实现方法研究及项目实际应用。

第一章介绍了分布式精扫系统的研究背景，回顾了传统的电信级系统的构建方式，分析了精扫系统的研究背景。对于移动互联网环境下全新的营销需求，传统的应用构建方式已经无法满足快速增长的营销需求，在全面地分析了国内外客户精确营销系统的开发和研究现状之后，针对互联网新环境下的电信精确营销需求，采用SOA服务治理技术方案。

第二章是分布式架构技术基础，在探讨分布式架构技术的起源、特性和主要实现方式的基础上，提出了一种采用SOA的服务化分布式架构的系统构建方法，这是一种将服务层与业务层进行分离的高效实现方法，能够明确系统的研究目标，降低模块间的耦合程度，模块间采用SOA的服务治理模式进行通信，各个子模块能够便利地进行维护和升级，架构的构建具有可扩展性、健壮性、稳定性和灵活性。

第三章是精扫系统需求分析和功能模块设计，在设计中重点考虑系统功能定位，给出了一种解决系统如何应对高并发、负载均衡及资源共享的实现方法。

第四章是系统架构设计，给出了系统的分层模型和软件模型。

第五章是精扫系统设计与实现，首先明确了业务功能模块，给出了接口设计规范，设计了数据库实体类，提出了一种基于基于Dubbo以及MQ的分布式服务调用的实现方法。

第六章是系统部署运行情况介绍，系统采用RabbitMQ搭建，采用Redis主从集群服务器模式，经测试运行表明，系统设计方法正确、稳定高效。

第七章对文章进行了工作回顾及对后续工作进行了展望。

# 第二章 分布式架构技术基础

2.1分布式架构技术起源与定义

随着Web2.0时代到来，各商业系统、企业级系统、互联网公司均面临着一个严峻的实际问题，即随着用户访问量的剧增、以及平台规模的不断扩大，原有的架构体系已经无法支撑与日俱增的业务需求[6]。系统升级需求，原有的垂直部署应用对于硬件需求极高、开发维护成本极其昂贵，系统平台升级扩展困难，容易出现单点故障灯诸多问题。

当前，企业级系统比较流行的技术架构基本主要以MVC架构为主，即Spring+ Struts2+Hibernate/iBatis+Tomcat，.NET MVC架构、EJB架构和LAMP架构也在部分内部系统中得到应用。各种主流MVC架构的实现细节存在差异，但是它们有一个共性，即垂直应用架构。垂直应用架构技术比较单一，学习成本低，开发上手较快，测试、部署和运维也比较简单，因此在很长一段时间里占据着统治地位。垂直应用架构如图2-1所示：



图2-1 垂直应用架构图

垂直应用架构面临的挑战:

在业务发展初期，应用规模比较小，基于JEE构建的垂直应用架构还能有效支撑业务的发展。随着业务的不断发展，应用规模日趋庞大，传统垂直架构开发模式的弊端变得越来越突出[7] [8]，具体有：

1、复杂应用的开发维护成本变高，部署效率逐渐降低，以实际基于MVC垂直架构开发的CRM系统为例，由于业务功能不断膨胀，最后代码全量编译和部署一次需要十几分钟。假若某个功能编译出错或者功能测试出问题，就需要重新打包编译，软件的部署效率极低。

2、团队协作效率差，部分公共功能重复开发，代码重复率极高，随着业务的发展，团队规模不断扩大，研发被分散到不同的开发小组中，不同的功能模块可能会依赖一些公共能力组件，由于没有类似服务化治理这种技术契约约束，也很难在不同团队间实现无缝沟通和共享，加之公共能力均为本地API调用，很可能重复开发API不能在团队中有效重用，会导致编写大量的重复代码。

3、系统可靠性变差。随着业务的发展，访问量逐渐攀升，网络流量、负载均衡、数据库连接等都面临巨大压力。某个节点的故障会导致分摊到其他节点的流量陡升，引起雪崩效应，高并发、大流量对于系统的可靠性要求非常高。垂直架构将所有的应用模块都部署到一个JVM进程中，如果某个应用接口发生故障，例如内存泄漏，会导致整个节点宕机。由于是对等集群部署，其余的对等节点也会出现类似问题，宕机可能会接连不断，严重影响业务运转

4、维护和升级困难。由于业务代码不断膨胀，功能越来越复杂，已有垂直架构模式下无法对复杂的业务进行拆分，代码间耦合度高，修改拆分困难，维护和定制都非常困难。

5、新功能上线周期变长，交付效率下降。（1）公共API变更导致测试工作量增加，当某个被重复开发的公共API发生变更后，会导致多出代码被动修改，重复测试，还会引入大量的回归测试工作量。（2）新功能无法独立部署和交付，新功能需要与老功能代码一起编译、打包、测试，如果测试出bug，整个系统需要重新打包和部署，强耦合会导致整个交付效率下降。最终原因是传统的垂直架构无法做到按照服务或者特性独立交付和运维。

分布式系统的演进与发展:

分布式系统是若干独立计算机的集合，这些计算机对于用户来说可视为单个相关系统[9]。这个定义中有两方面含义。第一个方面关于硬件：机器本身是独立的。第二个方面关于软件：对于用户来说他们就是在与单个系统打交道。

随着垂直架构的应用越来越无法满足大型系统与互联网应用的快速发展，大型分布式架构的系统应运而生，其演变过程，有一个显著的共同点，是在某个阶段网站的架构体系由垂直一体应用结构转化改造为服务化的体系，也就是常说的SOA，SOA系统之间以服务的方式来进行交互，这样保证了应用的可伸缩性、健壮性、高性能，对于一个庞大、多用户、高并发的应用系统而言这一点至关重要，所以实现分布式架构的重要一点是实现基本的子系统之间各种调用服务的请求与响应，除了这一点之外，对于访问量巨大的网站而言，主要都是采用可水平伸缩的集群方式来支撑巨大的访问量，这会涉及在服务交互时需要做负载均衡相关方面的处理，目前绝大多数分布式应用系统多数采用软件负载的方式来实现服务的交互，如何实现服务治理即常说的SOA是大型分布式应用系统的必备基础技术。

基于服务化主要是为了解决系统单点问题、垂直部署性能问题以及原系统功能扩展困难、由访问量剧增带来系统升级难度大等问题，但是数据量的增大还会带来更多复杂的问题，大型网站都是严重依赖缓存来提升性能的，数据量增长带来的效应就是单机会无法缓存所有的数据，需要引入分布式的缓存；数据量增长对于持久型的存储而言就更为复杂，通常会需要采用分库分表，引入NoSQL等方式来解决，对于模糊查询等需求更加复杂了，而对于电信级的系统而言，每天会产生大量用户业务数据、营销人员与客户交互的数据、用户使用产品过程中的行为等大量数据，这也就导致了随着访问量的增长，多数情况下用户产生的数据量也会暴涨，因此数据量增长带来的这些问题也是必须学会如何去解决的[10]。

2.2分布式架构特性与技术

分布式架构的重要特性更能显现其特点，其重要特性之一是：各种计算机之间的差别以及计算机之间的通信方式的差别对用户是隐藏的，同样，用户也看不到分布式系统的内部组织结构[11]。另一个重要特性是，用户和应用程序无论在何时何地都能够以一种一致和统一的方式与分布式系统进行交互。第三个重要特性，分布式架构的扩展或者升级是相对比较容易的，因为主要由独立的计算机集群组成，其中主要的公共能力、频繁调用的后端能力均由分布式服务实现，即使某个部分暂时发生故障，但其整体在通常情况下总是保持可用状态[12]。当某一个新功能或者API被替换或者升级，用户以及消费方程序不会察觉到哪些部分被替换或者维修，对于整体架构来说，部分服务的升级或者替换是透明、无影响的。

2.3分布式框架主要实现方式

RPC框架全称是Remote Process Call，即被称为远程过程调用，它应用广泛，实现方式有RMI、WebService等多种成熟方案。作为最早出现的分布式框架技术，其产生于垂直部署的应用将核心业务抽取出来，核心业务作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快适应多变的需求和升级要求。同时，将公共能力API抽取出来，作为独立的公共服务供其他调用者消费，以实现服务的共享和重用，降低开发和运维成本。应用拆分之后会按照模块独立部署，接口由调用本地API的方式演进为跨进程的远程方法调用，此时RPC框架逐步形成[13]。

单台服务器的处理能力受硬件成本的限制，不可能无限制地提升。RPC将原来的本地调用变为调用远端的服务器方法，给予系统处理能力和吞吐量近似于无限地提升可能与扩展能力，这是实现分布式服务的基础。

RPC的实现包括客户端和服务端，即服务的调用方与服务的提供方。服务调用方发送RPC请求到服务提供方，服务提供方根据调用方提供的参数执行请求方法，将执行结果返回给调用方，一次RPC调用完成。这其中涉及到调用参数以及响应结果的序列化和反序列化操作[14]。RPC调用过程如图2-2所示：



图2-2 RPC调用过程图

RPC框架实现的核心技术要点：

1、基于HTTP协议的系统间的RPC，具有使用灵活、实现便捷、开放并且天生支持异构平台之间的调用等多个优点。相对应的基于TCP协议的RPC实现版本，效率更高，但是实现起来更加复杂，并且由于协议和标准之间的不同，难以进行跨平台和企业间的便捷通信。

2、远程服务提供者需要以某种形式提供服务调用相关的信息，包括但不限于服务接口定义、数据结构、或者中间态的服务定义文件，例如Thrift的IDL文件，WS-RPC的WSDL文件定义，也可以是某种服务端的接口说明文档。服务调用者需要通过一定途径获取远程服务调用相关信息，例如服务端接口定义JAR包、获取服务端IDL文件等等。

3、远程代理对象：对于Java语言，它的实现就是JDK的动态代理，通过动态代理的拦截机制，将本地调用封装成远程服务调用。

4、通信：RPC框架与具体的协议无关，例如Spring的远程调用支持HTTP Invoke、RMI Invoke，MessagePack使用的是私有的二进制协议。

5、序列化：远程通信，需要将对象转换成二进制码流进行网络传输，不同的序列化框架，支持的数据类型、数据包大小、异常类型以及性能都不同。不同的RPC框架应用场景不同，因此技术选择也会存在很大差异。一些做得比较好的RPC框架，可以支持多种序列化方式。

SOA服务化架构技术要点：

在大规模服务化之前，应用一般是通过RPC框架，简单地暴露引用远程服务，通过配置服务的URL地址进行远程服务调用，路由则通过F5硬件或者SLB进行简单的负载均衡[15]。

当服务越来越多时，服务URL配置管理变得非常困难，F5等硬件负载均衡器的单点压力也较大。此时需要一个注册中心，动态注册和发现服务。第一点好处是，可使服务位置透明，服务提供者一般只要向中心地址注册服务接口名，服务框架就能动态对外发布，而消费者对于服务真实地址是无需关心的，消费者只需要配置好服务框架中心地址和调用方法，即可在本地缓存服务提供者列表，实现软负载均衡。SOA服务治理框架的第二点好处，随着服务间依赖关系变得错综复杂，甚至无法理清应用是依赖于哪些其他应用，应用之间的启停顺序，架构设计者都难以完整描述应用之间的调用关系。此时，一个良好的SOA服务跟踪系统可视化展示服务调用链，用于依赖分析、业务调用路径梳理等。

随着服务调用量的提升，服务的容量问题随即暴露，某些服务的机器支撑、网络配置等均需要进行汇总分析，此时需要采集服务调用KPI数据，进行汇总和分析，通过计算得出服务部署实例数和服务器的配置规格[16]。

在上述背景下，服务框架+服务治理的SOA服务化架构诞生了，SOA是一种粗粒度、松耦合的以服务为中心的架构，接口之间通过定义明确的协议和接口进行通信[17]。SOA需要架构设计者以及程序员站在一个新的高度理解企业级架构中各种组件的开发和部署形式，它可以帮助企业系统架构师以更迅速、可靠和可重用的形式规划整个业务系统。SOA能更加掌控企业系统集成和需求的快速变化。

SOA面向服务的一般原则：

1、服务可复用：不管是否存在即时复用的机会，服务均被设计为支持潜在的可复用。

2、服务共享一个标准契约：为了与服务提供者交互，消费者需要导入服务提供者标准契约，可以是IDL文件、Java接口定义、WSDL文件、接口说明文档。

3、服务松耦合：尽可能相对独立，尽量不依赖其他服务。

4、服务是底层逻辑抽象，对外底层逻辑不可见。

5、服务之间可组合、可整合。

6、服务自治：逻辑由服务所控制，独立位于清晰的边界内，服务已在边界内被控制，不依赖于其他服务。

7、服务是无状态的，这是重要的一点，服务应当不需要管理状态信息，因此能够维持松耦合与等幂性，服务应当尽可能设计成无状态，由专门的地方管理状态。

8、服务可被自动发现、更新升级无感知：服务发布上线后，允许被其他消费者自动发现，当服务下线时，消费者也可接收到下线通知。当服务功能升级更新后，消费者无变化、无影响。

SOA服务治理所解决的问题：

服务化之后，随着业务的不断发展，服务数量越来越多，应用服务化之后给系统运维带来很大的挑战[18]：

1、分布式框架下的服务调用性能。

2、服务化架构如何支持线性扩展。

3、如何实现高效、实时的服务多维度监控。

4、大规模分布式环境下的故障快速定界和定位。

5、分布式环境下海量日志在线检索、模糊查询。

6、服务的流控、超时控制、服务开降机等管控手段。

7、服务的划分原则，如何实现最大程度复用。

SOA服务治理能够较好地解决上述问题，SOA服务治理主要包括如下几个方面：

1、服务定义：SOA治理最基础的方面就是监视服务的创建过程。必须对服务进行标识，描述其功能，确定服务行为范围。

2、服务生命周期管理：服务需要进行规划、设计、实现、部署、维护，并最后下线。服务开发生命周期通常都具有五个主要的阶段。计划阶段、测试阶段、运行阶段、弃用阶段一一尽管该服务仍然在生产环境中运行，但是不会长时间运行，过一段时间将会下线，警告服务消费者尽快停止使用此服务；废弃阶段一一标识为服务的最后一个阶段，表示一个生命周期结束的服务。注册中心可以将该服务物理删除，也可以逻辑删除，标识该服务已经被废弃。

3、服务版本治理：服务发布后不久，服务提供者就开始根据需要对服务进行修改。包括问题修复，添加新的功能，有时还需要删除非必需的功能[19]。服务修改之后，需要对服务进行重新编译和打包，发布新版本的服务。新版本的前向兼容性，灰度发布等需要按照统一的策略进行管理。

4、服务注册中心：服务提供者如何发布服务？服务消费者如何订阅服务？已经发布的服务如何重新注册和下线，新注册的服务如何被消费者动态发现，需要统一的服务注册中心支持服务的订阅发布和动态发现机制。

5、服务监控:服务监控中心需要对服务的调用时延、成功率、吞吐率等数据进行实时采样和汇总，通过图形化报表的形式展示，以方便运维人员对服务的运行质量进行实时分析和掌握。

6、运行期动态质量保障:包括服务限流、服务迁入迁出、服务权限、服务负载均衡、超时控制等等[20]。通过动态治理，可以在不重启服务的前提下达到快速提升服务运行质量的目标。

7、服务治理提供快速的故障定界手段：其中快速定界主要包括两方面的内容。（1）大规模分布式环境下海量业务/平台日志的采集、汇总和实时在线检索，支持多维度的条件查询、模糊查询等，快速查看日志、方便问题定位。（2）分布式消息跟踪，通过调用链打通业务、服务调用和异常，发现线上系统故障源，通过大量分析得到相关系统参数指标，识别核心功能的服务调用依赖关系，评估可能的风险点、针对性改进以预防风险，同时为容量规划和扩容提供数据决策依据[21]。

8、服务安全：服务安全限制了几个方面，包括对于任意消费者调用服务的权限，敏感数据型服务对于所有用户的可见度，服务消费者和提供者数据交换过程中加密安全，服务的安全认证，内外防火墙和安全策略。服务治理需要能够提供关于安全性的功能设置，对于服务的访问进行权限控制，将服务的访问权仅限于授权使用者[22]。用户安全标识需要在服务调用中携带，用于授权敏感数据的访问。策略有几种，例如可通过动态生成令牌Token方式做安全访问，服务提供者动态生成令牌Token并告知服务注册中心，通过这种方式可以进行复杂的授权模型控制。

服务化的演变与治理的技术实现手段：

SOA架构中，服务消费者通过服务名称，在众多服务中找到要调用的服务的地址列表，成为服务的路由[23]。服务调用路由如图2-3所示：



图2-3 服务路由图

对于负载较高的服务来说，往往由相对应的多台服务器组成的集群实现，在请求到来时，为将请求均衡地分配到后端服务器，负载均衡程序将从服务对应的地址列表中，通过相应的均衡算法和规则，选取一台服务器进行访问，这就是服务的负载均衡[24] [25]。负载均衡算法如图2-4所示：



图2-4 负载均衡图

当服务的规模较小时，可以采用硬编码的方式将服务地址和配置写在代码中，通过编码的方式来解决服务的路由和负载均衡问题，也可以通过传统的硬件负载均衡设备如F5等，或者采用LVS或Nginx等软件解决方案，通过相关配置，来解决服务的路由和负载均衡问题。此时如果服务的机器数量在可控范围内，则维护成本能够接受。

当服务越来越多，规模越来越大时，对应的机器数量、集群规模也越来越大，单靠人工来管理和维护服务以及地址的配置信息，变得越来越困难。并且，依赖单一的硬件负载均衡设备或者使用LVS、Nginx等软件方案进行路由和负载均衡调度，单点故障的问题也开始凸显，一旦服务路由或者负载均衡服务器宕机，依赖它的所有服务均将失效。

为了解决上述问题，治理中心需要一个能够动态注册和获取服务信息的地方，来统一管理服务名称和其对应的服务器列表信息，称之为服务配置中心。服务提供者在启动时，将其提供的服务名称、服务器地址注册到服务配置中心，服务消费者通过服务配置中心来获得需要调用的服务的机器列表，通过相应的负载均衡算法，选取其中一台服务器进行调用[26]。当服务器宕机或者下线时，相应的机器需要能够动态地从服务配置中心里面移除，并通知相应的服务消费者，否则服务消费者就有可能因为调用到己经失效的服务而发生错误。在这个过程中，服务消费者只有在第一次调用服务时需要查询服务配置中心，然后将查询到的信息缓存到本地，后面的调用直接使用本地缓存的服务地址列表信息，而不需要重新发起请求到服务配置中心去获取相应的服务地址列表，直到服务的地址列表有变更（机器上线或者下线）。这种无中心化的结构解决了之前负载均衡设备所导致的单点故障问题，并且大大减轻了服务配置中心的压力。基于Zookeeper的注册中心部署，使用持久、非持久节点，通过集群间的zab协议使得服务配置信息能同步并保持一致，Zookeeper本身容错特性和leader选举机制，能保障方便地进行扩容。使用Dubbo服务框架并配合Zookeeper服务注册中心是目前本系统应用分布式思想构建后端服务的最佳实践[27]。如图2-5所示：



图2-5 Zookeeper服务注册图

2.4 采用SOA的服务化分布式架构实现系统构建

将集中式应用拆分为分布式：

大规模系统架构的设计一般原则就是尽可能地拆分，以达到更好地独立扩展与伸缩、更灵活的部署、更好的隔离和容错、更高的开发效率。具体拆分策略包括横向拆分和纵向拆分[28]。

1、纵向拆分：通过对业务进行梳理，根据业务的特性把应用拆开，不同的业务模块独立部署，对于本文讨论的精扫系统按照客户域、产品域、资源域、营销管理域等拆分，将复杂的业务线拆分成相对独立、灵活的具体能力域，由大变小、分而治之。纵向拆分方案示意图如图2-6所示：



图2-6 纵向拆分图

2、横向拆分：将核心的、公共的业务拆分出来，通过分布式服务框架将业务改造为服务化，消费者通过标准的接口规范来消费服务[29]。服务提供者独立开发、打包、部署、演进，与消费者解耦，服务相对于业务来说更类似于细粒度的组件，业务是一套完整的流程更加面向市场部门和客户。

使用Web服务或者RPC框架对业务进行拆分后，随着服务数的增多，需要一个服务治理框架，有效管控服务，提升服务运行期质量，提高整体系统框架的健壮性和可控[30]。服务治理主要解决了如下诉求：1）服务生命周期管理。 2）服务容量规划：通过集中式治理规划整体服务的机器支撑、机器配置、性能、时延指标、成功率、系统资源占有率等综合指标，为容量规划提供精确的数据支撑。3）运行期治理：业务发展高峰期或者压力峰值期，可对非核心服务采取降级、限流的措施，保证核心业务的运行稳定。分布式缓存失效时，系统压力转移到数据库，服务时延陡增，业务失败率升高，需要在线增大服务超时时间，保证服务成功率；非核心逻辑失败时，需要对业务执行本地降级逻辑。为了保证服务运行期质量，需要具备强大的在线动态服务治理能力。 4）服务安全：对用户进行分级、对敏感信息进行控制、对内部消费者和外部第三方消费者提供差异能力[31] [32]。

本文分布式框架主要技术选型：

1、Zookeeper服务配置中心：

基于ZooKeeper的持久和非持久节点，我们能够近乎实时地感知到后端服务器的状态（上线、下线、宕机）。通过集群间zab协议，使得服务配置信息能够保持一致。而ZooKeeper本身容错特性和leader选举机制，能保障我们方便地进行扩容。通过ZooKeeper来实现服务动态注册、机器上线与下线的动态感知，扩容方便，容错性好，且无中心化结构能够解决之前使用负载均衡设备所带来的单点故障问题，只有当配置信息更新时才会去Zookeeper上获取最新的服务地址列表，其他时候使用本地缓存即可。基于Zookeeper的服务配置中心的搭建，是本论文实现分布式服务dubbo接口的重要组成部分。

2、分布式服务框架Dubbo：

Dubbo是一款阿里巴巴开源的SOA服务治理方案的核心框架。

Dubbo功能架构如图2-7所示：



图2-7 Dubbo功能架构图

Dubbo提供了丰富的服务治理能力，其主要属性有：

（1）注册中心（Zookeeper）负责服务地址的注册与查找，相当于目录服务。服务提供者和消费者只在项目启动时与注册中心交互，注册中心不转发请求，没有负载压力。

（2）监控中心负责各服务调用次数统计，调用时间统计等，能以web页面等展示。

（3）消费者根据提供者服务命名空间向注册中心获取服务提供者地址列表，并会将地址列表缓存至本地，并根据负载算法直接调用提供者。

（4）注册中心、服务提供者、消费者三者之间均为长连接。

（5）注册中心与提供者之间是长连接，若服务提供者宕机，注册中心将立即推送事件通知消费者。

（6）注册中心若宕机，不影响已运行的提供者和消费者，因为消费者已缓存了提供者列表。

Dubbo提供的服务治理功能如图2-8所示：



图2-8 Dubbo提供服务治理图

3、分布式消息队列RabbitMQ&JMS

RabbitMQ 是由 LShift 提供的一个 Advanced Message Queuing Protocol (AMQP)的开源实现，由以高性能、健壮以及可伸缩性出名的 Erlang 写成，因此也是继承了这些优点。AMQP 里主要包含两个组件：Exchange 和 Queue。JMS(Java Messagese Service 即Java消息服务）本身是一组Java应用程序接口，它提供消息的创建、发送、接收、读取等一系列服务[33]。JMS定义了一组公共应用程序接口和相应的语法，类似于Java数据库的统一访问接口（JDBC），它是一种与厂商无关的API，使得Java程序能够与不同厂商的消息组件很好地进行通信。

JMS支持的消息类型包括简单文本（TextMcssage）、可序列化的对象（ObjectMessagc）、键值对（MapMessage）、字节流（BytesMessagc）、流（streamMessage)，以及无有效负载的消息（Message）等[34]。消息的发送是异步的，因此，消息的发布者发送完消息之后，不需要等待消息接收者立即响应，这样便提高了分布式系统协作的效率。

4、Memcache缓存

在高并发环境下，大量的读、写请求涌向数据库，磁盘的处理速度与内存显然不在一个数量级，从减轻数据库的压力和提高系统响应速度两个角度来考虑，一般都会在数据库之前加一层缓存。由于单台机器的内存资源和承载能力有限，并且如果大量使用本地缓存，也会使相同的数据被不同的节点存储多份，对内存资源造成较大的浪费，因此催生出了分布式缓存。

Memcache是一款开源的高性能的分布式内存对象缓存系统，最早是给LiveJ0umalZ提供服务的，后来逐渐被越来越多的大型网站所采用，用于在应用中减少对数据库的访问，提高应用的访问速度，并降低数据库的负载[35]。

为了在内存中提供数据的高速查找能力，memcache使用key-value形式存储和访问数据。在内存中维护一张巨大的HashTable，使得对数据查询的时间复杂度降低到O(l)，保证了对数据的高性能访问。内存的空间总是有限的，当内存没有更多的空间来存储新的数据时，memcache就会使用LRU(LeastRecentlyused）算法，将最近不常访问的数据淘汰掉，以腾出空间来存放新的数据。memcache存储支持的数据格式也是灵活多样的，通过对象的序列化机制，可以将更高层抽象的对象转换成为二进制数据，存储在缓存服务器中，当前端应用需要时，又可以通过二进制内容反序列化，将数据还原成原有对象。

5、k-v缓存Redis

Redis是一个高性能的key-value数据库，与其他很多key-value数据库的不同之处在于，Redis不仅支持简单的键值对类型的存储，还支持其他一系列丰富的数据存储结构，包括strings、hashs、lists、sets、sorted sets等，并在这些数据结构类型上定义了一套强大的API。通过定义不同的存储结构，Redis可以灵活地完成缓存、分布式锁等任务。

## 2.5 本章小结

本章主要介绍了主流的分布式架构技术，对分布式的起源和发展作了回顾。对目前主流的SOA服务治理模式及其原理作了详细的阐述。

另外，本章以开源SOA服务框架Dubbo为例介绍了业界主流的服务框架治理方案，对构建大型系统所需的分布式组件如MQ队列、分布式KV缓存等技术也作了详细介绍。

# 第三章 精扫系统需求分析与功能模块设计

3.1 精扫系统功能定位

目前，运营商竞争日趋激烈，产品体系日新月异，变化极快，在信息化的大背景下电信业市场已经全面转换为买房市场，运营商需要主动出击建立，主动迎合市场需求，对于探究客户需求、面向客户拓展业务的需求日益强烈，以往的消费群体也从单一化转向多样，从大众化的业务逐渐分化成精准化，个性化的消费群体[36]。

目前能够看到，越来越多的运营商主动将业务经营权下放到支局、社区，精扫系统就是运营商细分市场到网格之后，深化全区域营销体制改革的大背景下设计出来的支撑系统，它主要能够帮助商客支局、行客支局、客户经理、中小营业厅等在移动化营销场景下，主动向客户发起套餐推荐、业务办理、客户维系关怀等差异化服务和营销。它要解决的是客户群细分、面向社区以及细分行业、强化销售主动性、充分联系内外渠道等问题[37] [38]。精扫系统的建设目标是：对用户进行细分、差异化服务、实施主动营销，防止老用户流失问题，提高客户ARPU值以及经营效益。扩大与客户的接触面与深度，基于用户需求、市场竞争的营销方案制定、发布并实施相应的市场营销活动，并在执行过程中收集客户的有效信息，接受用户反馈、形成海量用户数据。利用电信分布广泛的营业厅、销售团队开展广泛营销活动，并对收集的海量业务数据进行营销分析，调整营销决策，形成闭环的营销过程管理体系，快速适应市场变化。基于上述目标，本系统的建设定位于以下主要关注点：对客户资料的全面收集整理、对经营决策的支持、对市场营销工作的支持、对社区、细分行业经营片区的精细化管理和渠道销售工作的支持、对日常管理和事务性工作的支持。

3.2 分布式框架在精扫系统中的应用

精扫系统整合了开源的分布式框架构建系统，采用了分布式服务框架Dubbo、基于Zookeeper的服务注册中心、分布式缓存memcache和redis、消息队列mq、Mysql读写分离、分布式存储FastDFS、页面静态化缓存等分布式技术，为精扫系统提供了全新的建设方式，各种系统与子应用之间通过分布式服务框架与消息队列进行通信，可以灵活迅速地构建系统和应用，系统也能更加灵活随着业务的改进而扩展，选择分布式服务框架以及相关的分布式技术是本文精扫系统的技术选型方案[39]。

3.2.1主动应对高并发

精扫系统在实际应用中，与以往传统的营业厅低流量的CRM系统不同，它面向的是分布广泛的支局、社区客户经理，每秒预计会处理的请求数会面临成千至上万的数量级，并发压力较大。如果采用传统的单点单模块后台应用来实现，压力极大，所有请求都要经过后台处理与响应，对机器计算能力、数据库负载等均要求很高，也极易出现应用的单点故障，稳定性与维护性都较低，不符合降低建设成本、提高效率的诉求[40]。分布式技术的应用能有效利用廉价PC Server，降低对高端主机的需求，降本增效，有利于减少维护硬件成本、减少软件扩容和开发开销。分布式可以发挥优势，将高并发计算请求水平分摊到低端机器上，同时多服务、多实例的部署有效提升系统稳定性。

3.2.2 可伸缩与负载均衡

在业务高峰期间，分布式服务可以针对某个热点应用或某个高频调用服务模块进行独立横向扩展，对热点问题进行集中处理，既不影响全局系统的稳定性，又能实现简单迅速扩容，充分体现易维护、易扩展的优势，也能积极应对应用的高并发、高峰值期间可能遇到的问题，实现资源的充分利用[41]。

除了请求数量时间上的分布不均衡以外，不同区域、不同用户群的访问数量也有不均衡性。通过Nginx或者F5等负载均衡技术，可将不同前端系统的大量请求，按照后台服务繁忙程度，将请求进行反向代理，使服务均衡调用，使得资源得到更有效地利用，减少浪费、提高系统处理能力。

3.2.3 资源共享

传统运营商级BSS系统包含有CRM系统、计费客户账务系统、购物车订单管理、销售品管理、渠道销售管理子系统、销售营业日报表系统等等若干系统，由于运营商IT建设进程历史原因，各个系统均都采用了独立建设模式，软件架构、开发模型、数据模型、业务流程、硬件设备等等方面均有差异，但又有相同点[42]。这一类独立开发的模式暴露出严重的问题，诸如设备利用率低下、投资效益差、各系统间资源调度难度大、扩展性需求响应差等问题。采用新的分布式开发模式，能有效克服这些IT建设问题，提倡资源共享、按需分配、应用服务化来建设新的IT支撑系统是为新系统奠定了良好的基础，为系统开发与成功上线以及未来的迭代升级打造良好的基因。

3.3 精扫系统功能模块设计

系统主要包含了系统管理、客户管理、客户评价、产品管理、营销管理、数据管理、查询统计、工作管理、考核评分等若干模块，精扫系统功能分解如表3-1所示：

表3-1 精扫系统功能表

|  |  |  |
| --- | --- | --- |
| 组成 | 功能 | |
| 前端入口 | 终端展示 产品办理 在线支付 | 客户服务 报表展示 客户信息管理 |
| 服务端前置层 | 过滤所有前端请求 转为内部标准接口 | |
| 客户信息管理 | 客户基础信息 客户接触意向录入 客户所属产品、帐户查询 | |
| 产品销售品管理 | 终端管理 电信产品管理 销售品管理 套餐管理 | |
| 订单管理 | 用户下单接口 订单管理 订单调度 营收算费 | |
| 市场营销 | 市场营销政策管理 活动管理 优惠管理 | |
| 报表数据分析 | 数据生成 报表定时任务 报表对外接口 | |

1、后台支撑系统模块：

包括系统管理（SYS）、客户信息管理（CIF）、产品（销售品上下架）管理、订单（销售管理、订单调度、支付、发票收据管理）管理、数据分析（客户数据、销售报表、营业数据分析）、市场营销（营销政策集中管理、销售推广方案管理、客户接触意向等等）、客户服务模块（各种查询 产品查询、余额查询、账户信息查询）

2、前端移动门户模块：

前端模块，采用H5移动端设计，并采取适配Android、IOS、微信公众号等方式，前端功能模块有：终端展示（包含终端、销售品、广告）、客户入口（客户信息采集、地理信息采集、客户接触意向收集、自动化客户分类查询展示）、销售办理模块（快速下单、快速交费续约、终端配送、流程查询、在线支付（微信、支付宝等）、电子发票收据）、客户服务入口（各类查询、账户信息、办理历史）、办公自动化（线索处理、发文指标处理、报表、会计对账）。

前后端功能划分：本系统采用了以上的前后端划分模式，前后端各自分布式化部署，后端支撑系统模块也进行服务化划分，对客户信息管理（CIF模块）、产品管理、订单购物车管理、营销自动化、客户服务模块进行服务化开发和部署。系统管理（SYS）主要是统一的管理入口，主要实现对工号、权限、组织机构、日志等方面的管理，其中的组织机构管理是该模块的核心，它集中式地管理全局的市场、区局、网格单元、客户经理的组织结构[43]。客户信息模块（CIF）主要针对我网存量用户、异网用户以及新增市场用户进行管理，并按照一定规则进行分类划分（地理信息、中高端、政企行业、个人等），也提供了按照规则分配给相应的客户经理的功能。产品管理模块（Product）主要实现的是对本地网所有终端、销售品、产品、业务的管理，上架下架，产品参数整理，该模块功能将以服务形式供前端移动门户模块调用，供业务人员在营销过程中使用。营销自动化（Market）模块，本模块依据市场营销政策、产品卖点、销售推广方案为主导，为客户经理提供最核心的销售细节支撑，并提供客户接触信息管理、订单派单等功能，也将以服务化的形式供移动门户模块调用。客户服务管理提供各种查询和展示功能。后台数据分析模块与周边各系统接口以及数据库读库进行交互，统一汇总并整理出全量数据以及报表。

前端移动门户模块，是一个以Html5、原生Native JavaScript为主要技术开发的移动前端web层，采用流行的Js库和框架设计开发的单页web应用，与后端通过大量Http通信进行交互，它是本系统的移动化办公解决方案，移动门户系统自身轻量、快速，主要功能依赖系统后端提供的各种分布式服务[44]。终端产品展示功能主要调用了后端产品管理模块功能；客户入口层主要调用了客户信息模块（CIF）；销售办理模块调用了后端的销售管理以及营销模块；客户服务层也调用了后端的客户服务模块。

## 3.4 本章小结

本章分析了精扫系统的具体需求和定位，阐述扫系统主要的功能和所面向的用户。在明确了功能需求之后，分析了分布式框架和服务治理技术与系统开发相结合的优势和特点，突出了技术层面在系统整体解决方案上的规划和设计。

在梳理清系统功能需求之后，对系统功能模块进行了具体的划分，系统整体采用前后端分层的模型，在系统后端结合了SOA服务治理和分布式开发技术，在后端的功能设计中，对可复用、功能有通用性的模块进行了横向划分，独立出多个项目模块，为接下来系统的整体架构设计打下基础。

# 第四章 精扫系统架构设计

精扫系统按功能划分为前后端两大系统功能块，后端系统按照功能又划分了若干服务化模块与一个统一管理（SYS）块。除此之外，在前端与后端之间设计了一层服务端前置层，为前后端通信和数据加工提供服务。集中治理后端服务之间通讯的主要技术是Dubbo分布式服务框架、Zookeeper服务注册中心、RabbitMq消息队列，这三者是整个后端系统运行的分布式基础。Dubbo框架除了提供了模块间互相同步调用的接口，也自带了服务治理、全局监控与调度的功能，可以实时监控整个后端各系统之间接口调用运行的情况，Zookeeper注册中心则负责动态实现服务注册、动态同步服务信息、服务负载均衡等功能。

4.1 精扫系统结构与数据模型

精扫系统采用分布式服务框架、通过集群化部署的各分布式中间件将庞大的整体系统分拆成多个服务实例，将这些服务均匀地部署在多个服务器上并行运行，最终供前端移动门户调用，系统运行的过程中服务调用能有效的负载均衡（采用Nginx反向代理或者KeepLive等负载均衡技术），服务器资源可以按需调用、系统按需扩展且具有弹性、服务故障也能通过dubboManager及时监控，各模块迭代升级快速、维护简便[45]。精扫系统逻辑架构如图4-1所示：



图4-1 精扫系统逻辑架构图

客户精扫系统的整体业务架构包括运营商级别下游系统、系统总控层及前置平台与入口组成，各模块相互关系如图4-2所示：



图4-2 客户精扫系统整体业务架构图

4.1.1 精扫系统分层模型

客户精扫系统分为移动门户前端、服务端前置层（内外网络访问、防火墙、负载均衡、参数适配）、服务层（包含模块有“系统总控层、客户域、产品域、订单域、营销域、统计报表域”）、分布式服务平台层（服务注册、调用、服务治理、消息队列、分布式文件处理），持久化层（读写分离mysql数据库、缓存）。

各层功能如下：

1、前端移动门户层与后台服务端的交互经过了服务端前置层，该层隔离了内外网络环境，将前台http请求接收并对相关敏感字段进行加密后，加工为Dubbo消息同步调用后端服务[46]。服务端前置层起到了服务调用隔离、信息安全控制、网关控制的功能，并能针对前端的请求进行业务分发和流量控制，并有效解耦了业务调用协议，将不同的协议融合到一起，使调用者和服务提供者对协议差异性保持透明化，简化了开发难度，提高了系统稳定性和可扩展性。服务端前置既有以往的企业总线的一部分功能，又有自己独立的功能特征。

2、在服务端前置的转发下，通过分布式服务平台层，服务调用将根据请求调用到各个相应模块中，各服务模块都位于服务层中，分布式服务平台层是基于统一部署的开源分布式系统构建的，集群化部署了Dubbo分布式服务框架、Zookeeper注册中心、RabbitMQ消息队列、Kafka消息队列、FastDFS分布式文件存储系统，分布式服务层对外提供了完备的开发接口，包括Dubbo服务发布，Dubbo服务获取，MQ队列注册获取，Zookeeper的调用（主要应用于服务注册、zk分布式锁、服务器竞争选举等场景）、分布式文件系统存取[47]。

3、服务层以分布式服务平台为基础，开发相关的系统、业务逻辑，服务层各项业务均独立部署并发布，服务层各项组件在分布式平台的基础上构成了整个精扫系统的后端服务。

4、持久化层包括了数据库服务器、Redis缓存、文件服务器等。数据库集群以Mysql数据库为基础进行读写分离部署，Redis缓存也采用集群部署。Mysql数据库采用读写分离的形式进行部署，为关键应用分配写库资源，报表等非实时应用分配写库资源。Redis缓存支持KV键值对以及各类数据结构的存储，也提供了相应的持久化功能，能很好地支持频繁的数据读写操作。服务层的应用均可方便的调用Redis缓存来加速对持久化资源的存取。

客户精扫系统分层关系如图4-3所示。



图4-3客户精扫系统分层模型

4.1.2 精扫系统软件模块

客户精扫后端系统从软件模块划分的角度分成了系统总控模块、客户域模块、产品域模块、订单域模块、营销域模块、统计报表域模块，各模块的持久化是与持久层进行交互完成的。

系统总控模块支持全局的精扫系统工号管理、组织机构管理、后台配置产品（对接产品域）、后台手工配置客户信息（对接客户域）、订单手工调度、营销策略管理（对接营销域模块）、按周期定时生成管理统计报表并支持导出等功能，整合了对后端系统的各项控制。

前端移动门户层与后端服务的主要业务交互有：

1、各项业务、产品查询调用。

2、产品下单，订单生成后消息报文由前端层发送至前置，再由前置转发到订单域。

3、客户信息收集，营销意向管理，需要与营销域模块、客户域模块进行交互。

4、前端客户经理、支局查看报表，与后端统计报表域进行交互。

4.2 系统数据模型设计

精扫系统基于现阶段运营商市场营销的需求，从业务与系统开发的角度针对客户、产品、市场营销三个数据域进行了细化和建模，提出了系统的总体核心概念模型和逻辑模型。数据域与相应的服务层组件划分是密切联系的[48]。

以下定义了系统的各主题域，包含了重要的业务概念和彼此的关系，以及定义了重要业务实体例如产品、客户、帐务、订单、营销主题、客户接触、客户问题反馈等等以及内在关系。在主题域的业务实体定义之后更细化一步，集中阐述和解释业务问题、明确各个数据域具体内容，把业务逻辑和交互方式详细描述。

1、客户主题数据：

三户核心块 -- 描述核心的三户模型 由客户、用户、帐户组成，客户由运营商面向的客户的社会信息为基础构建，用户体现了业务域的信息，帐户体现的是资金域的信息。

客户信息块 -- 对客户基本资料和统一视图提供数据支撑

客户接触块 -- 对客户的地理位置、行业信息、客户画像、标签提供统一接触和数据落地、支撑服务。

客户评价块 -- 对客户信用、积分提供支撑。

客户关系块 -- 对客户之间关系提供支撑。

维系与挽留 -- 对客户忠诚度的维护涉及到的计划和活动提供数据支撑，对有高危离网倾向用户提供维系支撑。

2、产品主题数据：

产品管理块 -- 对产品的生命周期和配置管理以及相关功能提供数据支撑。

订购实例块 -- 对产品的订购提供支撑。

业务资源块 -- 对业务资源的管理提供支撑。

套餐、终端管理块 -- 对销售品打包套餐提供数据模型支撑，对移动终端、宽带终端提供支撑。

3、营销市场主题数据：

销售渠道块 -- 对支局、代理商、客户经理提供数据支撑。

市场活动块 -- 对市场营销活动管理提供数据支撑。

市场营销计划块 -- 对市场营销计划管理提供数据支撑。

销售机会管理块 -- 对销售活动、客户意向提供数据支撑。

4、订单主题数据：

订单生成块 -- 对订单生成、订单流转流程提供数据支撑.

订单管理块 -- 对订单内销售品的查看、添加、销售品参数、套餐打包提供支撑。

订单处理块 -- 后台对订单管理提供支撑

订单调度块 -- 将订单调度到相应的运营商CRM系统，并实现自动化受理，对调度流程提供数据支撑。

客户、用户、账户与订购产品之间的关系逻辑如图4-4所示。



图4-4 三户模型与产品关系图

5、报表域：

报表域主要由报表分类、报表表结构等数据模型构成，并提供数据支撑。主要包含的报表类型有：

重点指标发展报表 -- 3G、4G新用户发展报表、宽带发展报表、融合套餐发展报表。

重点收入报表 -- 每日营收报表、当月累计营收报表、往月计费+营收总计报表，可基于组织机构、地区、客户属性、产品类型、套餐类型、帐期等属性进行分类统计查询。

客户维系报表 -- 欠费客户清单统计、客户套餐迁移统计、离网风险客户统计等。

## 4.3 本章小结

本章为系统开发进行了架构设计。首先，对整个精扫系统的作了详细的分层，分别从逻辑架构角度和业务角度分别规划设计了架构模型。其次，明确了前后端分离的分层结构，确定了前端与后端进行交互所使用的通信协议，确定了后端所需要的SOA服务框架。

在此基础上，对系统的数据模型进行了分类和设计，确定了客户、产品、订单、市场营销的数据模型，并且每一个主题域的数据模型都分别与后端的多个独立模块相对应。建立起功能和数据这两个系统所需的构建基础。

# 第五章 精扫系统功能设计与实现

5.1 业务功能设计

业务系统功能设计，主要清晰地划分了前后端服务层的各项业务，以及如何通过具体编码的设计实现业务逻辑。客户模块域、产品域、订单域等等，均由业务层次进行划分，在开发技术层面上，我们对各划分的模块域进行分块开发，各自成立独立的服务并独立部署在集群中，形成分布式化的后端系统[49]。本节中，对各模块的功能进行描述，并对编码提供具体设计思路。

5.1.1 前端模块

前端模块也称为移动门户端，是整个系统的应用入口，是精扫系统面向使用者的客户端，系统所设计的服务层在前端移动门户层均有入口[50]。技术上，实现方式是Html5页面+Js框架+Ajax通信的方式实现的web级应用。功能重点设计了移动H5端。前端模块可由手机浏览器端、微信端进行访问，后期二期迭代可规划ios端与Android端app功能。

主要入口：

1、产品展示层 终端展示、融合套餐展示、流量包展示、IPTV展示、宽带套餐展示。

2、客户入口 对接客户新增、客户信息采集、客户标签管理等功能，主要对客户的社会信息维度、电信级用户维度进行信息采集和管理。

3、快速办理下单 包括直接下单、套餐办理、套餐升级、收银台界面、单产品办理、意向采集等功能，快速办理可由产品展示界面直接接入，也可以单独进入对应模块办理。

4、订单管理 包括客户历史订单，客户经理全量订单查询等。

5、客户服务 对接客户服务界面，可以对客户的相关电信产品进行所有查询和历史追溯，同时包含余额查询、充值缴费、续约续期等功能。

6、办公入口 主要由客户经理和销售渠道使用。对接客户维系挽留，定向接受上游系统派单，对高危离网标签的客户进行查看和信息管理，以便更进一步提供维系客户的策略。办公入口同时还具有报表查看、指标分析筛选等相关功能。

5.1.2 系统总控层

系统总控层是整个精扫系统的后端入口，以web界面为展示，实现整体的系统管理、系统数据管理功能，对后端开放了统一的可视化后台界面。开发技术选型主要以Spring+ SpringMVC+Mybatis+bootstrap为构建基础，SSM是功能完善的Javaweb级开发选型，bootstrap是Web页面的主要框架[50]。项目的构建采用Maven管理和编译打包。SYS模块功能设计如下：

1、工号权限配置

系统提供精扫系统的工号管理功能，如：工号增加、工号删除、工号修改、密码修改等功能外，应实现系统的功能点与数据查询范围的分别授权管理。数据查询范围应对应组织机构授权，以适应员工调动后权限所在范围的自动调整。为进一步简化功能授权，本模块设计了角色管理功能，角色管理分为功能角色、数据角色。具体角色的维护由系统管理员完成，功能角色可分为客户经理、片区渠道管理员、营销中心经理、地市渠道管理员、地市考核人员等。数据角色可分为集团级客户管理人员、商居客户管理人员、渠道管理人员等。为应对系统管理问题，还设计了区公司、地市分公司两级管理员以方便将来的系统维护、管理。

2、组织机构配置

组织机构的管理是核心管理功能之一，系统要提供按层级组织机构的设置、管理功能，并以树型目录展示。为适应员工调动及区域营销中心管辖业务范围变化频繁的实际情况，并为日后系统维护分级管理奠定基础，系统中组织机构框架与叶子子元素可以分别设置。组织架构是指系统中设置并能与实际业务管理中市场、部门以及细分营销单元对应的业务管理组织[51]。叶子元素是指系统中设置并能与实际业务管理中具体的客户经理、后台员工、渠道营业厅、代办点等对应的业务管理元素。组织机构由系统管理员管理，终端节点由区局和营销单元维护。

3、营销团队管理配置

系统提供了营销团队的管理组合功能，对区局、代理商等各渠道均可组合建立营销团队，营销团队是一种临时的营销单元。营销团队可以作为一个营销主体承接营销派单任务，执行具体的营销任务。客户经理或者销售人员可以同时隶属于两支以上的营销团队，并可按照团队进行任务分配，指标分配。

4、产品、套餐、业务参数配置

产品管理包含了：对运营商产品域参数模型的管理，对本系统的产品的上下架进行配置，可配置的产品域相关层次包含了：产品层次、售品层次和终端层次。

（1）产品层次的概念有 移动通信、固网通信、宽带、专线业务、云计算业务、翼支付业务（互联网理财）、企业短彩信等等。

（2）销售品层次包含了 周期性套餐业务、打包套餐业务等等，销售品层次是整个面向客户营销的系统基础，市场营销的核心。

（3）终端层次包含了 手机终端、宽带终端、IPTV终端的配置、参数管理功能。

套餐管理层次上，主要功能针对套餐信息、套餐目录进行管理：

（1）定期同步CRM系统中的主要资费套餐并予以展示，系统维护人员针对套餐资费信息进行补充完善并再次发布，方便由客户经理在前端中进行查看了解并向客户推荐。主要配置的信息有套餐名称、套餐ID、网络信息、计费信息、销售对象、主要销售渠道、上下架时间、套餐有效时间、可叠加套餐、互斥套餐等等。

（2）系统提供分级配置功能，一级配置为产品级别，第二级下属的为套餐级别，第三级下属的为可叠加资费。

5、业务参数配置

（1）业务规则层，哪些套餐、产品适用于哪些客户，优惠资费的办理条件等等，主要配置模型针对产品服务模块（PRODUCT），可对每一项产品、套餐订购配置相应的规则

（2）帐目参数管理，本系统基于市场营销的适用性，对运营商CRM系统的原始帐目进行了归并和整合。帐目整合为一级帐目和二级帐目，一级帐目针对收入大类进行归并，二级帐目对收费细项进行归并，方便后期对客户的信息展示以及销售人员客户经理的报表查看。

6、营销主题参数配置

对市场部、政企、区局制定的营销活动进行主题配置，集中配置到营销数据域。后期客户经理、代理渠道商等在使用精扫系统发展业务过程中，可以在前端系统中直观地展示近期热门的营销活动，对营销策略、市场政策有快速明晰的感受，能够更快更好地展开营销业务活动。

营销主题管理也是本系统的核心之一，它将客户域、产品域的数据整合起来，提供了业务指引，是业务发展的核心模块。

7、订单后台控制、流程配置

对精扫系统发展的用户、订购的订单等，提供后台控制的功能，提供订单实时调度的能力。提供流程库控制定制功能，可配置订单从生成、添加产品、计算金额、订单生成后总览、订单后台审核、订到流转CRM进行自动受理等等步骤，实现自动化计算、自动化受理的初步功能。

5.1.3 客户服务模块

客户模块简称CIF(Customer Info)，是纯后台服务模块，接受系统控制层的配置接口发起的调用，对外开放Dubbo接口供其他模块调用服务，并接受相关消息队列。服务调用对接了系统总控层、服务端前置层、订单管理服务层、报表统计层等各层的调用。

客户域是所有业务的基础，所以在构建思路上CIF是一个相对更加底层的服务模块，它将作为所有业务的基础进行服务。

内部技术选型，Spring+Mybatis作为主要技术选型，采用Maven作为项目构建工具，Jekins作为持续集成的工具。

1、客户资料管理

客户资料管理是构建客户信息视图的基础，客户服务模块在这个目标上建立了基本的客户信息增删改查的功能，在以客户为中心的角度，将统一客户信息分为客户基本信息、服务信息和扩展信息。

客户基本信息包括：客户信息、用户信息、帐户信息。其中客户信息包含客户识别信息、身份信息等；用户信息包括用户基本信息、订购信息等；帐户信息包括帐户基本信息、帐务关系、账本信息、余额、累积量等等。

服务信息用于管理与客户接触、渠道服务的信息记录：包括接触信息、服务请求信息、受理信息、退缴费信息。

扩展信息是客户挖掘、客户等级健康度、精确营销的信息，通过客户经理的深入接触、计费分析以及营销服务的过程中，分析并统一收集管理的、包括客户个性化、中高端、消费行为、成本、营销信息、合同、客户规模重要度等等方面并预留扩展模型。

客户的本质是电信产品的使用者，包括个人以及组织，潜在异网客户也在客户管理范围内。帐户指客户使用产品的付费实体。用户是电信公司订购产品的实例，每一个已订购的产品实例必定会一一对应到一个用户，也即一个服务实例对象。一个电信客户可以拥有多个用户实例。

本网客户、新增客户、异网客户三种可相互转化。转化过程就是精扫系统的业务实现过程，将非本网用户转化为本网用户，业务和营销就发生在这一过程中，成功营销客户使用本网产品即为转化为本网用户，同时关联新增了本网产品实例。客户关系相互转化如图5-1所示。



图5-1 客户转化图

2、帐户信息管理

帐户信息管理主要负责对接计费账务系统，实时同步计费信息、话单历史、帐务信息、套餐余额、使用量。

3、用户信息管理

用户信息对接CRM系统，实时同步每一个电信客户下的用户实例，并对接到用户下关联的详细产品信息、套餐信息。

5.1.4 产品服务模块

产品服务模块同样是纯后台服务模块，是以纯后端服务实现的方式来构建的基础模块，以分布式框架为底层进行部署。也采用了Spring+Mybatis等相关JavaEE技术进行开发。

产品层也是具有通用的底层特性，它能够为各种业务提供服务，对接移动门户层以及订购管理层，主要提供Dubbo服务的同步调用以及RabbitMQ的异步处理。划分了产品、销售品、终端、Connector等多个子层。

1、产品信息管理子层

对接CRM系统实时同步产品信息，对接系统SYS层，主要实现运营商基础产品的配置，增删改查功能。提供产品订购的调用服务。

2、销售品信息管理子层

对接CRM系统实时同步销售品信息，对接系统SYS层，实现本系统内销售品、资费套餐的上下架配置、生失效规则配置、打包销售品的配置（实现捆绑销售、模块化销售），下游数据持久化对接到数据库层。

3、终端信息管理子层

管理手机终端、宽带终端、IPTV终端，对硬件编号、出入库实现统一管理。对接各个服务的标准调用。

业务流程逻辑主要为：根据前端移动门户层发起的产品订购，查询到最近的终端库存和渠道。客户经理点击确认并发起终端订购动作后，前置收到包含终端出库的消息，同步调用终端信息管理模块，对终端的出入库实现订购时同步出库，终端出库成功后异步发送结算报文给下游终端结算系统，实现终端的快速出入库，高效实现终端管理、结算，也简化了客户经理的业务发展流程。终端订购出库动作如图5-2所示。

5.1.5 订单管理服务模块

订单管理服务模块是业务的直接对象方，是重点的业务核心模块，所有的新用户订购电信产品、老用户订购叠加套餐、续费、套餐迁移等等一系列的业务动作都会直接对接到订单管理服务模块，可以说，订单管理模块是业务发起中心。

具体业务发起流程为：客户经理在前端移动门户层发起订购动作，前端业务界面会逐个统计订购的套餐等等，当客户经理为客户订购好产品后，提交购物车，前端层会发起Ajax请求至前置，前置将消息进行转发，转发给订购管理模块，本模块进行业务逻辑校验、资费合并计算、套餐内容展示等各项业务处理计算，过程中会逐一调用客户模块、产品模块等基础模块，对客户信息、业务资费等进行调用供业务处理。



图5-2 终端订购出库流程图

订购管理模块处理完毕后，会实时返回前端，展示订购结果。客户经理确认调用无误后，最终确认并调度到收银台界面。前端收银台界面汇总订单处理模块报告的营收的费用并展示，收银台同时可对接翼支付、支付宝、微信端的支付功能，可在线收款。收银成功，最终也调用后端前置功能模块，再由前置发起对订单管理模块的请求。订单管理模块对接下游电信计费系统发起营收请求，请求成功后返回收银成功消息。流程到此，一个完整的订购动作成功完成。订购动作如图5-3所示。



图5-3 订购动作流程图

1、订单人工处理功能：

订单处理模块还包含了其他若干功能，其中对接SYS模块，主要设计了订单人工处理介入的接口。

2、订单查询、订单流转阶段展示、订单处理：

人工干预订单，订单金额修改、订单作废、订单重新生成、订单手工派单、订单手工算费等等功能。

3、订单调度、自动流转CRM进行自动受理：

订单调度功能主要设计为初始默认订单的流转过程，初始化默认阶段，订单在营收款结算成功后，会自动转为下游CRM受理、资料入集团CRM系统。待CRM受理成功后，会返回给订单管理模块，此时订单管理模块会自动反馈给前端移动门户层，提示客户经理和客户产品订购成功并受理完毕，订单状态也会进行同步更新。

订单调度功能额外地扩展了订单流转流程，可以在SYS模块中进行流程配置后，选择自主订单流转模式，实现个性化订单受理过程。

5.1.6 市场营销模块

营销模块(简称Market)是市场活动的需求方、发起方、计划政策制定方，统筹指引全局的业务，在市场营销制定的背景下，才能有目标有计划地开展核心业务。营销模块是上层服务模块，以纯后台应用方式构建，开放Dubbo接口接收SYS模块的调用进行营销配置，对组织机构、人员等进行市场政策配置和活动派单，前端移动门户层调用市场营销模块服务，获取营销流程等信息，开展活动[52]。

1、营销流程定制功能：

营销流程定制是一个较为笼统的概念，其中一个完整的流程涉及到电信运营商全方位的营销体系和过程。设计一个营销流程，营销活动是本质主体，市场活动评估、营销方案、市场调查、品牌、产品、广告均围绕营销活动主体展开，是配套设施。市场营销逻辑关系如图5-4所示。



图5-4 市场营销关系图

2、发放营销计划与派单营销：

营销流程制定完成，需用来指导客户经理怎样对营销单制定的目标客户进行营销的说明性内容，包括派单方式、营销具体流程、营销单类型、营销具体目标客户、营销具体执行的竞争区域、码号资源总览、终端资源总览、定制的营销套餐和资费、营销活动资费特征、营销话术脚本、营销单级别、目标客户群。

营销流程可进行手动执行派单或者依据流程进行自动流转到相应部门，类型包括有业务发展、客户维系关怀、欠费催缴、产品升级等等。目标客户群、宣传脚本等内容通过SYS管理页面进行导入。

3、客户接触反馈管理、营销后客户意向信息收集：

本功能是设计实现执行市场营销过程中，与客户接触的反馈管理，也同时采集客户意向、获取市场活动评价，形成营销活动闭环。

5.1.7 报表统计模块

统计报表模块是纯后台项目构建，实现移动门户层、SYS层对数据的统计分析需求，该模块统计完整的业务发展数据。显示和展示均由前端系统完成。

1、客户数据报表：

客户维度报表数据包括：新入网客户、用户资料、欠费客户、高值客户，生成相应报表。另外，针对客户行业、客户行为、客户属性等进行多维度标签设计，在此基础上统计报表支持从多维度进行统计分析。

2、业务发展报表：

业务发展报表包括每日3G、4G用户数据 、宽带发展业务报表、按周期统计收入等报表。能够对应组织机构提供分不同层级的收入、欠费、业务发展、用户规模、派单等报表。能够对照客户资料、用户分帐期等提供欠费明细、按发展人渠道等维度划分提供查询导出等功能。

主要功能点设计包含如下几点：

报表生成功能可以区县公司、政企、渠道、营销中心、支局、客户经理、代理商、门店等发展渠道作为销售维度，以客户性质、客户等级、行业类型、客户行业、客户状态、中高端、客户画像标签、具体产品类别等作为业务维度，来具体细分地进行各项报表的统计查询。

功能还包含对在网客户状态进行区别划分，对高质客户、低ARPU客户、低质量客户、高危客户等多维度进行划分和统计，掌握在网用户的信息变化趋势。

其他还包含对收入情况的统计，按帐期、指标；包含对欠费、高额消费预警的欠费明细情况统计；包含对派单查询统计的功能；包含对时间段统计各渠道业务发展情况的汇总统计功能。

对各级别机构的查询报表功能设置相关权限。各机构只能通过SYS系统或者前端系统查看到自己及下属机构或者叶子节点的业务报表。各客户经理只能看到自己以及所在组的业务发展情况。

5.2项目接口设计规范

1、Dubbo接口规范设计：

Dubbo的工作过程中，主要有四种角色：Provider：暴露服务方称为“服务提供者”；Consumer：调用远程服务方称为“服务消费者”；Registry：服务注册与发现的中心目录服务称之为“服务注册中心”；Monitor：统计服务的调用次数和调用时间的日志服务称之为“服务监控中心”[53]。

Dubbo工作原理：

（1）注册中心负责服务地址的注册与查找，相当于目录服务，服务提供者和消费者只在启动时刻与注册中心交互，注册中心不转发请求，服务方与消费方直接进行对接和调用。

（2）Monitor模块作为监控负责统计各服务调用次数、时长等。

（3）服务提供者在项目启动后会向注册中心注册其提供的服务，并汇报调用时间到监控中心。

（4）服务消费者向注册中心获取服务提供者的地址列表，并根据负载均衡算法直接调用提供者，同时汇报调用时间到监控中心。

（5）注册中心、服务提供者、消费者三者之间均为长连接，注册中心通过长连接心跳检查感知服务提供者的存在，服务提供者若宕机，注册中心将立即推送事件通知消费者。

（6）注册中心和监控中心若宕机，不影响已正常运行的提供者与消费者，消费者本地会缓存提供者列表。

当服务方发布服务后，只需要将含有service接口的jar包发布到maven私服上，由消费者拉取并依赖即可调用，消费者调用Dubbo接口就像调用本地服务一样。

Dubbo服务方接口设计时使用序列化的Request作为通用请求类作为请求入口参数。使用序列化的Response类作为通用返回类作为返回规范参数。标准的Resquest和Response类如图5-5 所示。





图5-5 Request、Response标准请求、返回参数类

详细说明：本系统针对请求和返回参数制定了规范，其中Request、Response类是各模块之间交互的通用请求、返回参数类。Request类中，tracelogId定义了日志跟踪id，serviceCode定义了业务场景划分后，具体的服务编码，消费者对特定接口调用时指定serviceCode会调用到特定服务处理类。params为具体的请求参数map容器，具体的请求参数以键值对的形式放置其中再请求到服务方。Response中，success为调用标识，responseCode为返回码，服务方预先会定义好返回码及其含义文档，供前端调用方知晓，responseMsg为返回信息。消费者通过success标识判断调用成功与否，泛型T由服务方提供，具体T类会定义调用的返回值JavaBean类。对外开放的接口以Java接口形式给出定义，如下例：

public interface IMarketService {  
 Response doMarketingService(Request request);

Response doCustomerService(Request request);

...  
}

接口定义中返回值与入参均采用标准规范类，定义了方法调用名。服务接口尽可能大粒度，每个服务方法应代表一大类功能，而不是某功能的一个步骤，否则将面临分布式事务问题，Dubbo暂未提供分布式事务支持。本系统中服务接口以业务场景为单位划分，并对相近业务做了抽象。相近业务的划分以Request类中的serviceCode进行区分，每一个serviceCode对应一个细分的完整事务服务。

Dubbo服务的对外发布过程，将接口方法与请求、返回类、泛型T类进行打包，并发布到私服。实际生产调用时，由服务方向注册中心注册，消费方无需提前通知即可在注册中心中根据接口注册信息找到服务并进行调用。

2、RabbitMQ消息接口规范设计：

MQ全称为Message Queue, 消息队列（MQ）是一种应用程序对应用程序的通信方法（消息传递），一般用作进程间通讯。MQ的主要应用场景是异步处理的需求，例如后端对客户资料大并发请求的同步流程、客户信息采集流程、产品信息批量刷新流程、客户新办套餐资费异步施工生效等流程均可使用MQ队列作为技术选型[54]。

MQ的优势是本身是异步的，往队列里发送消息后无需等待，不同于通信协议。如HTTP协议（同步），客户端发出请求后必须等待服务器回应。MQ的适用场景为高并发应用来不及处理、实时性要求不高以及多应用之间异步通信，且耗时操作。

RabbitMQ是由Erlang语言开发，实现Advanced Message Queuing Protocol (AMQP高级消息队列协议)的消息中间件标准。消息中间件主要用于组件之间的解耦，消息的发送者无需知道消息使用者的存在，反之亦然。

本系统RabbitMQ设计规范，采用Spring框架与RabbitMQ结合的开发方式，在Spring配置文件中建立RabbitMQ的配置文件。MQ消息发送采用JSON格式化字符串方式，规范如图5-6所示。



图5-6 RabbitMQ队列示意图

Spring中使用MQ队列模板类（RabbitTemplate）进行开发，作为生产者向mq队列发送消息具体开发流程如下图5-7 所示。



图5-7 生产者向队列发送消息开发流程

消息messageStr就是参数map进行JSON格式化后的字符串。调用配置好的RabbitMQ模板类send方法进行消息发送，消息会自动发到队列中供消费方消费。消费方从队列中取得msg再通过JSON反序列化得到参数map，以此继续执行业务逻辑。

3、HTTP请求规范：

REST(Representational State Transfer)是一种编程风格，特别是针对网络应用的设计和开发，采用REST可以显著降低开发的复杂性，提高系统的可伸缩性。REST强调使用统一的规则来规范对资源的操作。前端移动门户层请求服务端前置规定均采用Http请求，需符合Restful风格。

（1）RESTful规范中，每个url代表一种资源（resource），url中不含有动词，只能有名词，

（2）主要采用的HttpMethod有：GET方法，用于查询获取资源、POST方法，用于创建资源、PUT方法，用于修改资源、DELETE方法，用于删除资源。URL由网络协议(HTTP、HTTPS)、服务器地址、版本、接口名称、参数列表构成。在URL中加入版本号version能够保留旧服务的兼容性留出空间以升级接口。示例：

标识版本号为v1的交易接口：

GET <https://www.telemarket.com/v1/trades>

创建一条订单信息:

POST /v1/order HTTP/1.1

Host: [www.telemarket.com](http://www.telemarket.com)

（3）复杂的数据报表查询请求采用JSON报文封装请求参数，并以POST方式发送。

5.3数据实体类设计

5.3.1 客户域数据模型

客户域数据包含客户数据Customer，客户数据是精扫系统面向的客户数据核心模型。围绕客户数据，本系统建立了一套完整的客户数据体系，包括关联的客户认证信息、用户数据、帐户数据、用户帐本数据。各数据实体分析如下：

（1）客户Customer是具体的电信客户对象

（2）客户认证信息Customer\_Identify定义了客户的身份信息

（3）用户User是具体的电信产品使用者

（4）帐户Account记录了用户的产品计费信息

（5）帐本记录了用户帐户下具体的余额信息和账务信息。

相关数据实体关系如图5-8所示：



图5-8 客户域数据

5.3.2 产品域数据模型

产品域定义了精扫系统主要营销的产品实体对象。产品域包含了具体的电信产品规格、具体的套餐规格、套餐实例以及套餐所包含的产品实例。

（1）套餐offer表，定义了套餐的具体规格。

（2）套餐实例表offer\_instance，凡是订购了电信的套餐并生效的均会产生一个套餐实例，实例表记录了电信用户所有的生效实例。

（3）产品表product定义了现有的电信产品规格信息。

（4）套餐—产品关系表offer\_product，定义了套餐组合中所包含的产品成员。

产品域数据关系如图5-9所示：



图5-9 产品域数据

5.3.3 订单域数据模型

订单域定义了精扫系统的具体业务动作。订单域由客户经理为客户进行下单后产生业务订购数据。

（1）订单order主表记录了业务订购后的具体订单信息。

（2）订单历史表order\_history是以往订单信息，为用户查询和后续订购提供依据。

（3）订单—产品实例关联表，记录了订单中关联的产品信息，订单中的产品信息是订单信息中的具体订购对象。

订单域数据关系如图5-10所示：



图5-10 订单域数据图

5.4 基于Dubbo以及MQ的分布式服务调用实现

本节主要阐述精扫系统tele-market的具体开发方案及实例，其中主要以市场营销服务模块marketing-project项目为代码实例。市场营销模块定制了营销策略和优惠方案。

5.4.1实际项目工程文件结构

市场营销模块工程结构图如图5-11所示。

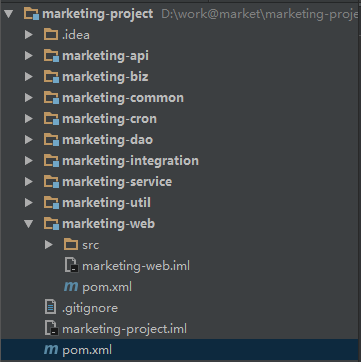


图5-11工程文件结构图

最外层marketing-project工程是整个项目的父工程，项目根目录下的pom文件是聚合pom文件，整合了整个项目的依赖和项目子模块module结构。

分层结构与实例代码：

marketing-web块是项目的主入口module。其子pom.xml文件定义了本module以war包构建。marketing-web是整个项目调用入口，connector包下主要由Dubbo接口和RabbitMQ接口为主。Dubbo接口对外发布服务，RabbitMQ接口消费订单Order系统送来的消费队列。Web模块源代码文件结构图如图5-12所示：

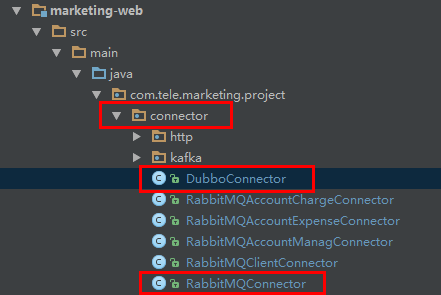


图5-12 web模块文件结构图

marketing-api是Dubbo对外接口jar包module，只定义接口方法、入参、返回值，具体实现均由marketing-web中DubboConnector类定义。如图5-13所示。

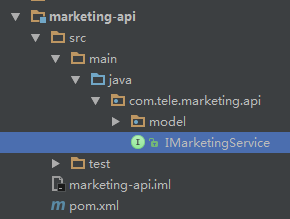


图5-13 marketing-api模块文件结构图

marketing-biz层是业务handler层，该module结构如图5-14所示：

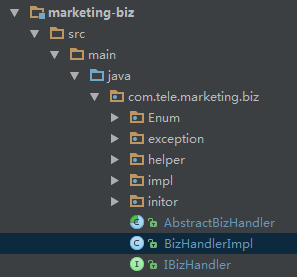


图5-14 biz模块文件结构图

当marketing-web入口获得外部调用之后，经过参数处理、业务分类后，会进一步调用biz层进行具体的业务处理。Marketing-biz层接收到web层的调用后，会根据web层处理后传来的参数和serviceCode具体判断应该调用哪个具体handler类来处理业务[55]。将具体业务handler类与serviceCode进行Map绑定是实现handler类注册的具体实践，负责handler与serviceCode注册的类就是AbstractBizHandler类，BizHandlerImpl继承了该类且是实际Handler处理类，BizHandlerImpl继承了AbstractBizHandler之后可直接访问注册容器serviceHandlers。AbstractBizHandler具体代码编写流程如下图5-15所示。



图5-15 AbstractBizHandler业务Handler注册流程

以上代码编写流程是serviceCode注册的主要部分，他的主要思路是Map容器注册，在Spring容器实例启动时提前实例化，Spring容器运行时serviceCode已注册完毕并可供BizHandlerImpl调用。BizHandlerImpl是biz层主逻辑类，起到接收web层发来的消息并根据具体的serviceCode注册信息分发调用具体handler。

marketing-service层，主要是service组件，主要起到service复用和事务控制作用[56]。由此层向下至调用Dao层，均为标准MVC开发模式。以上就是一个后端服务项目构建的主要方案。主要涉及的技术栈是Maven+Spring+Mybatis+Dubbo+RabbitMQ+Redis。

5.4.2项目Dubbo实现

Spring项目中Dubbo开发具体步骤如下：

1、第一步是将Dubbo开发环境与Spring进行集成。

（1）主pom文件中，集成Dubbo、zkclient依赖包。

<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>dubbo</artifactId>  
 <exclusions>  
 <exclusion>  
 <artifactId>spring</artifactId>  
 <groupId>org.springframework</groupId>  
 </exclusion>  
 </exclusions>  
 </dependency>  
 <dependency>  
 <groupId>com.github.sgroschupf</groupId>  
 <artifactId>zkclient</artifactId>  
 <exclusions>  
 <exclusion>  
 <artifactId>log4j</artifactId>  
 <groupId>log4j</groupId>  
 </exclusion>  
 </exclusions>  
 </dependency>

（2）marketing-web模块是对外服务模块，在该module下的resource包中配置dubbo-provider.xml：

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns=<http://www.springframework.org/schema/beans>

xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.2.xsd  
 <http://code.alibabatech.com/schema/dubbo>

http://code.alibabatech.com/schema/dubbo/dubbo.xsd">  
</beans>

2、开发环境配置好后，构建项目子Module模块：marketing-api，创建供远程调用的接口IMarketingService，同模块下定义包model，定义Request、Response方法，制定标准的远程调用实体类。IMarketingService代码参考Dubbo接口规范设计标准编写。对外Dubbo接口结构如图5-16所示：

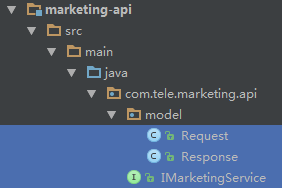


图5-16 对外Dubbo接口结构图

3、构建子module模块：marketing-web，创建connector包，包下定义DubboConnector类，该类具体实现第一步定义的IMarketingService接口，包括参数基本校验、日志打印、加工为标准的RequestObject入参对象等工作。DubboConnector代码编写流程如下图5-17所示：



图5-17 DubboConnector代码编写流程

同时，编写完成DubboConnector实现了IMarketingSerivce接口后，将该接口配置添加到上一步中创建的dubbo-provider.xml文件中：

<dubbo:application name="${marketing\_application}" />  
 <dubbo:registry id="zookeeper\_service" protocol="zookeeper" address="${zookeeper\_service\_url}" />  
 <dubbo:protocol name="dubbo" port="${dubbo\_service\_export\_port}" />  
 <dubbo:service interface="com.tele.marketing.api.IMarketingService" ref="dubboConnector" registry="zookeeper\_service" />

4、添加marketing-biz模块，创建第二步中依赖的bizHandler对象的接口IBizHandler，创建AbstractBizHandler这个抽象类，以实现各个handler类的注册。至此可在各handler中开发具体的业务代码。通过api、web、biz三层结构，清晰地完成了内部服务业务代码向外部开放为Dubbo接口的过程。

5、外部模块调用过程：

后端其他的项目需要调用marketing-project的dubbo接口时，只需在所需要引用的包下配置与Spring集成的xml配置文件dubbo-consumer.xml,配置文件结构如图5-18所示。

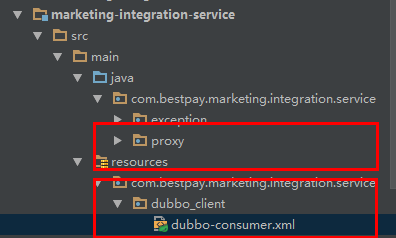


图5-18 引用外部接口文件结构图

该consumer.xml定义了消费方需要消费的接口名，以下代码中清晰地引入了tele系统下部署的CIF客户信息管理项目的对外接口，分别配置引用了账户查询accountQueryService、销账accountConsumeService、客户信息查询customerQueryService等接口。引用的xml配置信息如下：

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.springframework.org/schema/beans"  
 xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.2.xsd  
 <http://code.alibabatech.com/schema/dubbo>

http://code.alibabatech.com/schema/dubbo/dubbo.xsd">  
 <dubbo:annotation/>  
 <dubbo:consumer check="false" />  
 <dubbo:reference id="accountQueryService"  
 interface="com.tele.customer.manager.AccountQueryService"  
 retries="0" timeout="5000"/>  
 <dubbo:reference id="accountConsumeService"  
 interface="com.tele.customer.service.AccountConsumeService"  
 retries="0" timeout="5000" />   
 <dubbo:reference id="customerQueryService"  
 interface="com.tele.customer.manager.CustomerQueryService"  
 retries="0" timeout="5000"/>   
 </beans>

dubbo-consumer xml中引入的接口Spring容器在启动后，会自动加载并创建实例供程序代码依赖调用。对外引用服务的代码一般都集成起来，形成proxy包，集中管理引用外部接口的代码，形成proxy代理类。

下面以proxy包中调用CIF项目查询客户信息的接口为例，首先在proxy包中创建CifCusProdFacadeServiceProxy类，类中引用Dubbo接口CustomerProdFacade。调用方式和普通依赖注入一致。调用过程如下图5-19所示。



图5-19 调用cif的dubbo消费者接口流程图

当编写好CifCusProdFacadeServiceProxy代理类后，我们的marketing-biz业务层以及marketing-service层就可以方 便进行Dubbo调用了。当我们的各个项目以war包的形式部署完毕后，就可以根据程序中配置的dubbo配置信息进行各服务之间的分布式调用，真正实现了完整的分布式开发及部署。

5.5基于Zookeeper的服务注册中心设计

通过Zookeeper来实现服务动态注册、机器上线与下线的动态感知，扩容方便，容错性好，并且无中心化结构，能够解决之前使用负载均衡设备所带来的单点故障问题，只有当配置信息更新时才会去Zookeeper上获取最新的服务地址列表，其他时候，应用需要调取Dubbo服务时只要使用本地缓存的信息即可[57]。

Zookeeper可以有效地管理服务订阅/发布，有效避免硬编码地址信息，能将服务统一管理起来，可以有效地优化内部应用对服务发布/使用的流程和管理，使用Zookeeper构建的服务注册中心就是专门用来管理服务订阅/发布的配置管理节点。

Zookeeper诞生于Apache Hadoop的一个子项目，它一开始的设计目的主要是用来解决分布式应用中经常遇到的一些数据管理问题，主要针对的是服务器集群关系信息、统一命名服务、状态同步信息、分布式应用统一配置等等。

本系统使用的Dubbo分布式服务框架建议使用dubbo-2.3.3以上版本的zookeeper注册中心客户端。生产环境即可部署Zookeeper作为注册中心。Dubbo本身未对Zookeeper服务器端做任何侵入修改，只需安装原生的Zookeeper服务器即可，应用程序本身所需的配置仅仅只是在dubbo的生产者和消费者配置文件中定义所需要注册的zk地址。所有注册中心逻辑适配都在调用Zookeeper客户端时完成。开发时，本系统使用Linux发行版Ubuntu来完成Zookeeper运行环境的搭建。

对于开发调试阶段，使用单机模式即已够用，主要步骤是搭建zk环境，之后再搭建Dubbo-admin管理环境并将dubbo-admin管理环境的注册地址指向我们已搭建好的zk地址。以下是dubbo与zk整合的搭建过程：

（1）进入/user目录，使用wget命令从官网下载Zookeeper。下载完毕后解压，拷贝配置文件zoo.cfg:

sudo wget http://www.apache.org/dist//zookeeper/zookeeper-3.3.3/zookeeper-3.3.3.tar.gz

tar zxvf zookeeper-3.3.3.tar.gz

cd zookeeper-3.3.3

cp conf/zoo\_sample.cfg conf/zoo.cfg

（2）执行vim zoo.cfg，将配置文件修改为如下内容：

#内容

tickTime=2000

initLimit=10

syncLimit=5

dataDir=/home/dubbo/zookeeper-3.3.3/data #数据目录

clientPort=2181

dataLogDir = /tmp/Zookeeper/log #日志目录

（3）创建配置中的相应目录。

cd /tmp

mkdir ZooKeeper

cd ZooKeeper

mkdir log

mkdir data

（4）最终启动zk，至此Zookeeper服务注册中心搭建完成:

cd /usr/ZooKeeper/bin

./zkServer.sh start

（5）配置dubbo-admin的管理页面，方便管理页面。

先下载dubbo-admin-2.4.1.war包，在Linux的tomcat部署，先把dubbo-admin-2.4.1放在tomcat的webapps/ROOT下，然后进行解压：jar-xvf dubbo-admin-2.4.1.war。

在webapps/ROOT/WEB-INF下，有一个dubbo.properties文件，在其中配置成指向Zookeeper，使用的是Zookeeper的注册中心，配置示例如下：



然后，启动Tomcat服务，用户名和密码：root,并访问服务，显示登陆页面，说明dubbo-admin部署成功，并且成功将注册中心关联到我们已搭建好的Zookeeper服务器上。至此Dubbo与Zookeeper整合环境已搭建成功，在我们实际开发项目过程中，要将本机开发的对外Dubbo服务接口注册到zk服务注册中心上，只需要定义明确的zk注册地址即可。实际示例如下：

<dubbo:application name="tele\_marketing\_application" />  
<dubbo:registry id="zookeeper\_service" protocol="zookeeper"

address="zookeeper://127.0.0.1:2181" />  
<dubbo:protocol name="dubbo" port="2181" />  
<dubbo:service interface="com.tele.marketing.api.IMarketingService"

ref="dubboConnector" registry="zookeeper\_service" />

当我们开发的服务提供者provider上线后，在Dubbo-admin管理页面上就能看到接口成功注册的信息。

## 5.6项目RabbitMQ实现

整套分布式系统，在运行时，如果遇到大数据量并发请求或者对时效性要求较低的场景时需要使用异步处理机制。RabbitMQ是一个优秀的异步队列框架，也采用分布式的集群部署模式。RabbitMQ在开发时，使用Spring框架进行集成，一般需要先配置MQ监听container，需要设定host、username、password、工厂类等等相关定义，这些配置可确定一个connection-factory。配置完成后，需要继续定义listener-container监听具体队列，需同时定义接收队列消息的Connector类。具体xml配置信息如下所示：

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:rabbit="http://www.springframework.org/schema/rabbit"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd  
 <http://www.springframework.org/schema/context>

http://www.springframework.org/schema/context/spring-context.xsd  
 <http://www.springframework.org/schema/rabbit>

http://www.springframework.org/schema/rabbit/spring-rabbit.xsd">  
 <bean id="marketRefConnectionFactory"

class="com.rabbitmq.client.ConnectionFactory">  
 <property name="requestedHeartbeat" value="240" />  
 <property name="connectionTimeout" value="2000" />  
 </bean>  
 <rabbit:connection-factory id="marketRabbitmqConnectionFactory"

host="${rabbit.host}" port="${rabbit.port}"  
 username="market" password="market#2015."

virtual-host="${rabbit.market.vhost}"

connection-factory="marketRefConnectionFactory" />  
 <!-- 读取 market 用户下的队列 -->  
 <rabbit:listener-container

connection-factory="marketRabbitmqConnectionFactory"

acknowledge="auto"  
 prefetch="1"

concurrency="${channel.concurrency}"

max-concurrency="${channel.max-concurrency}">  
 <rabbit:listener queue-names="q\_rebate\_order\_trans"

ref="rabbitMQOrderConnector" />

</rabbit:listener-container>

其中rabbitMQOrderConnector定义了处理队列消息的，rabbitMQOrderConnector类实现了MessageListener接口，MessageListener接口由Spring制定标准接口，接口中定义标准接收消息方法onMessage()，处理消息代码编写流程如下图5-20所示。



图5-20 RabbitMQ监听接口开发流程图

onMessage()方法中，具体处理了Message对象，以UTF-8编码格式转为Json字符串，并将Json字符串再转换为请求对象rebateWebgateParam，之后再调用bizHandler.doBiz方法转入biz层进行业务逻辑编写。至此完整地实现了RabbitMQ在项目中的实际使用。

5.7 分布式缓存Redis工具类设计

Redis本身是一个高性能的Key-Value内存数据库，与其他很多k-v数据库所不同的地方在于，Redis不仅支持简单的键值对类型的存储，还支持很多丰富的数据结构，包括Strings、hashs、lists、sets、sorted、sets等等，并在这些数据类型上构建了一套强大的API供调用。本章节主要阐述如何在项目中集成Redis、如何使用分布式缓存大大提高系统的运行效率、并极大减轻关系型数据库的压力。

Spring项目中，如果要使用Redis，第一步依然是进行Redis环境配置，首先将Redis的Java客户端Jedis引入maven工程环境依赖中。接下来在Spring的配置xml文件中定义Jedis连接池和JedisCluster配置：

<!-- Jedis 连接池配置-->  
<bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig">  
 <property name="maxTotal" value="${redis.maxTotal}"/>  
 <property name="maxIdle" value="${redis.maxIdle}"/>  
 <property name="minIdle" value="${redis.minIdle}"/>  
 <property name="testOnBorrow" value="true" />  
</bean>

<!-- jedisCluster 配置 -->  
<bean id="jedisCluster" class="redis.clients.jedis.JedisCluster">  
 <constructor-arg index="0">  
 <set>  
 <bean class="redis.clients.jedis.HostAndPort">  
 <constructor-arg index="0" type="java.lang.String" value="${redis.cluster.host1}"/>  
 <constructor-arg index="1" type="int" value="${redis.cluster.port1}"/>  
 </bean>  
 <bean class="redis.clients.jedis.HostAndPort">  
 <constructor-arg index="0" type="java.lang.String" value="${redis.cluster.host2}"/>  
 <constructor-arg index="1" type="int" value="${redis.cluster.port2}"/>  
 </bean>

</set>  
</constructor-arg>

</bean>  
</beans>

配置完毕后，Spring容器启动之后就会装载jedisCluster实例，我们就可以在开发过程使用jedisCluster实例并封装实际所需要的工具类了。封装Jedis工具类的过程很简单，也是使用依赖注入JedisCluster，并将常用的CRUD方法封装为同一的方法提供调用，集成Redis工具类的具体流程如图5-21所示。



图5-21 集成redis工具类方法流程

业务代码中，一般使用Redis的场景主要为，当有频繁的需要读写操作的热点数据，（1）第一次的查询可以使用先从关系型数据库中读出，并缓存到Redis中并设定失效时间，下一次查询请求到来时，若Redis中缓存了所要查询的键key，则可以直接在Redis中取得值并返回结果，如果所要查询的键值不存在则视作第一次查询，查询DataBase再同步Redis后返回。

（2）如果有update操作，则按照键update value后，同步到数据库中再返回。采用Redis会极大提升读的效率，减轻数据库读的压力。对于时效要求不高的数据则不进行update操作，只设置相应的缓存失效时间，失效过后再次查询会刷新为最新的值。实际开发编码示例如图5-22所示：

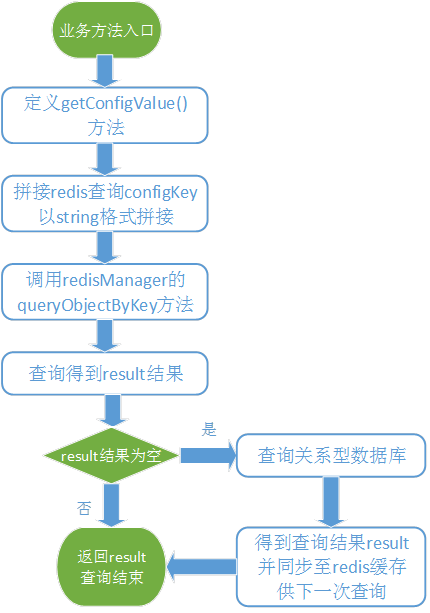


图5-22 实际使用redis工具类进行查询操作

上面是market市场营销模块使用redis进行查询的示例过程，该示例使用firstBizMappingCache进行了redis查询的封装，在其中依赖了RedisManager工具类。firstBizMappingCache.getMarketConfigValue(configKey)调用的是RedisManager的redis查询方法，入参configKey是查询条件拼接的string字符串，redis中规定key值唯一，因此编写代码时我们一般可以使用“预定义的常量头+查询参数”拼接成查询key进行insert和get操作。

5.8服务端前置层Front-Platform项目结构和设计方案

服务端前置是本系统连接前端移动门户入口和后端各服务项目的桥梁，它承接了分离前后端的任务，让前端专注于展示业务、专注设计与用户客户进行交互、屏蔽复杂的业务逻辑；也让后端各服务项目能以大的业务角度来分离代码和进行模块、分布式化开发，后端开发也只需要关心具体业务逻辑代码的编写，无需去考虑具体的展示和请求消息。服务端前置层使得分布式开发部署后，业务可充分解耦，消息协议与调用依赖更加清晰明确，前端与后端项目在后期迭代过程中都可以方便快速地进行升级和部署，减少依赖复杂性，提高系统健壮性、扩展性[58]。

Front-Platform项目，统一封装了所有内部系统的对外能力（或接口），对外提供服务。服务端前置不关心业务内容，它只是业务信息的搬运工。设计架构如图5-23所示。



图5-23 服务端开放平台分层图

Front-Platform项目，内部架构分为API网关层、业务转发层、Dubbo向下接入层、协议权限管理中心等module。功能如图5-24所示。



图5-24 Front-Platform项目内部架构功能图

服务端前置实现“一次接入，全程无忧”，为接入的业务提供如下重点功能：

1、Http报文防篡改：支持可选对称和非对称加密方式，对全报文进行签名加密验证；

2、Dubbo接口转换：简单配置参数实现dubbo接口，返回值转换为HTTP服务；

3、版本管理： 实现同一业务接口不同版本同时服务，可实现特定版本的开关操作；

4、动态限流：按业务系统和接口维度做流量监控限流，保障大流量业务期间业务平稳运行；

5、服务升降级：按需求对接口服务进行动态调整，实现服务升降级；

前后端系统接入方式采用Http方式和Dubbo方式：

1、前端请求方式：

接入后对外服务的地址接口名称规范： /系统名称/api/接口方法/接口版本， 如/product/api/queryList/v1。

完整路径为：<https://mapi.telemarket.com.cn/product/api/queryList/v1>。

2、请求参数格（POST）,content字段中以json格式存放业务参数键值对：

{“channeled”:“调用渠道",

“userInfo”:“用户信息",

“reqTime”:“发起时间”,

“token”:“令牌”,

“signType”:“验签方式",

“sign”:“签名值“,

“content”: “业务参数”}

3、后端业务系统相应格式：

业务系统按照公司约定的response对象返回至服务端前置，后端业务系统相应格式规范如下：

Response规范:{“success": true,

“result”: {业务返回},

“responseCode”:“标准返回码",

“responseMsg”: “返回信息" }。

其中，前置将result返回值透传至外部调用方，responseCode按照架构组统一规划进行定义，responseMsg为对该调用情况的文字说明。

## 5.9 本章小结

本章从细节层面对精扫系统的构建作了研究和设计实现。首先对业务功能进行了具体设计，对各个模块的功能和互相之间的关系做了详细设计。在着手开发之前，对系统全局的接口开发规范做了定义。

项目的构建开发完全基于SOA服务化治理思想，先从Dubbo实现原理角度阐明了Dubbo是如何对接口调用的双方进行服务的发布、注册以及调用的。针对服务注册中心，本节引入了zookeeper在Dubbo服务治理中的实践方案，zookeeper作为Dubbo的注册中心这一角色，承担了服务提供者的服务发布、服务消费者根据xml配置文件连接注册中心获取服务信息、以长连接形式感知服务的生命周期并实时通知服务消费者等功能。此外，对Dubbo提供的Dubbo服务治理的Dubbo-admin还进行了部署和相关测试，对Dubbo的服务治理功能做了详细分析和相关代码实践。

本章对本系统的创新点之一，服务端前置架构做了详细设计，服务端前置隔离了前端客户交互层与后端服务化模块层之间交互的复杂性，使得前端web端与后端服务层等各模块独立开发、独立部署，实现业务逻辑与用户交互的解耦，从架构层面真正意义上实现分布式化开发部署。服务端前置还实现了统一服务接入，无论是APP端还是浏览器端访问精扫系统均依赖于统一的中间层接口，所有的接口指向服务端前置网关，由服务端前置完成复杂的通讯协议适配、数据封装、安全、监控、日志、限流等相关功能，使前后端的接口调用实现极简化、智能路由的功能。服务端前置根据具体请求信息，将处理请求转化为Dubbo协议请求，并根据路由转发到后端各服务模块。真正实现了降低耦合性，减少了通用接口的重复开发，前端层可面向统一接口进行开发，实现快速高效迭代。

# 第六章 系统测试

系统测试时，首先按在实际运行环境进行部署，部署时使用Linux服务器(centos 6.4 64位)，采用Nginx负载均衡技术、搭建消息队列系统RabbitMQ，安装Redis集群部署存储系统。

6.1 系统测试部署

系统测试时，需要部署的组件除了Linux、MySQL、Tomcat外，还包括RabbitMQ、Redis、Nginx等环境部署。系统实际部署运行架构如图6-1所示。

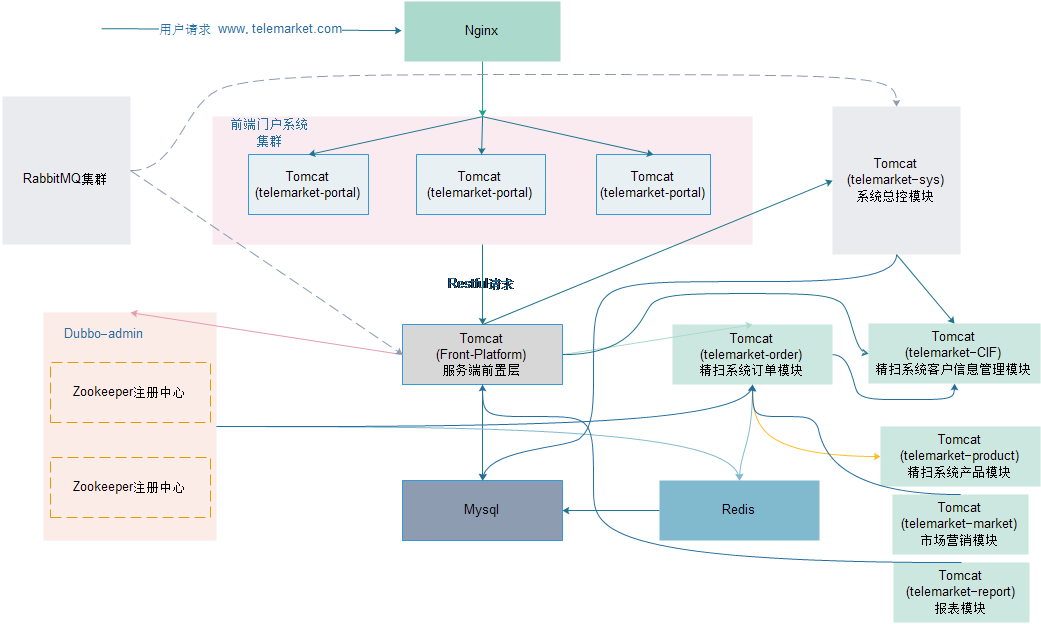


图6-1 系统部署运行架构原理图

系统的测试运行流程如下：

1、用户在移动web端登录精扫系统，经Nginx反向代理和lvs负载均衡，请求会转发到前端门户项目。

2、前端门户项目名为telemarket-portal，前端门户的后台采用Springmvc开发，使用tomcat服务器集群部署。前端门户接收到用户的请求经过前端业务处理和封装，将Restful请求发送到服务端前置。

3、服务端前置(Front-Platform)使用tomcat服务器部署，接收到前端消息后经过消息封装和过滤转发Dubbo调用到后端各服务模块。

4、精扫系统后端各模块使用Tomcat服务器进行独立部署，各模块启动时会自动到Zookeeper服务器上进行Dubbo服务注册，同时会开启监听RabbitMQ队列消息服务。后端各模块经过接收服务端前置转发的Dubbo调用进行相关的业务处理。

5、后端各模块与Redis集群和Mysql数据库连接完成持久化。

6.2 RabbitMQ测试

RabbitMQ集群节点包括磁盘节点和内存节点。磁盘节点将数据存放在磁盘，内存节点将所有数据存放在内存，但有例外的情况是持久的queue的持久内容将被保存到磁盘上。在一个RabbitMQ集群中可以共享 user，vhost，queue，exchange等，所有的状态和数据在所有节点上均可以复制。内存节点虽然不写入磁盘，但是它执行比磁盘节点要好。

本系统测试RabbitMQ性能的方法如下：

首先声明了3个具有不同属性的消息队列queue；接着向exchangeA发送消息，这些消息具有persistent属性。最后，创建3个消费者分别从上述3个消息队列queue中获取消息。在RabbitMq2.7.0版本中，提供了部分UI插件，可用来监控RabbitMq中的clientconnect、exchange 、vhost、queue、binding 、发送速率、接收速率等，如图6-2所示。

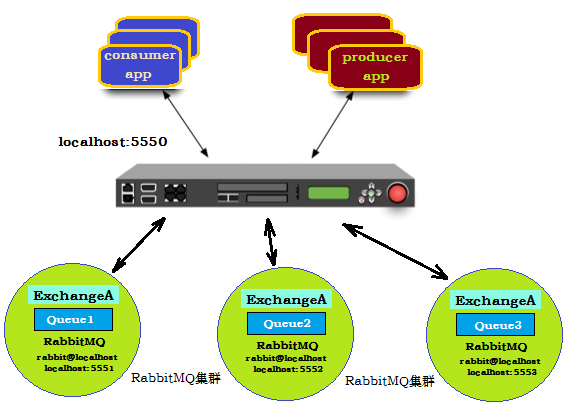


图6-2RabbitMQ性能部署图

客户端使用负载服务器发送消息， 使用rabbitMq管理端可以看到集群镜像模式中队列状态，如图6-3所示。

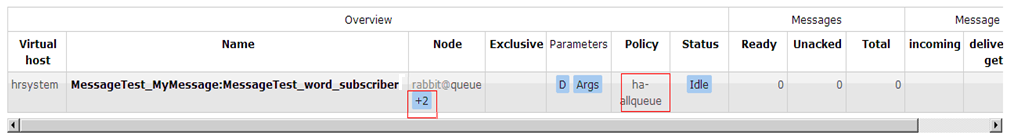


图6-3 rabbitMq性能测试图

测试结果表明：

1、消息队列会被复制到所有节点，这时任何节点如果出现故障都不会影响正常使用某个队列。

2、在100MB网络中，使用RabbitMQ集群，生产者使用异步模式发送数据比同步模式快18倍。发送速率高达8790msg/s，接收者速率高达6823msg/s，CPU占用为7%-11%。表明系统资源占用明显降低，数据通信速率显著提高。

3、消费者使用二线程接收数据时性能最佳，如果数据处理[程序](http://www.xuebuyuan.com/)含有复杂的处理逻辑，则应该增加几个线程接收数据。

4、集群中持久化的数据保存在客户端所连接的节点，其它节点都会对这些数据进行镜像，这样可保证各节点队列中的数据相一致。

6.3 Redis主从集群系统性能测试

1、 主从集群架构部署

Redis主从集群架构部署如图6-3所示。这种架构的优点是避免Redis单点故障，实现读写分离，满足读多写少的应用场景。

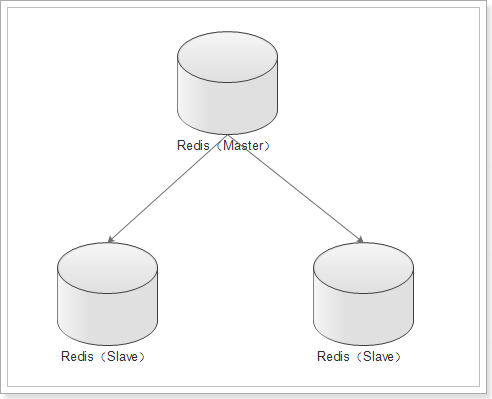


图6-4 redis主从部署图

首先创建6379、6380、6381目录，分别将安装目录下的redis.conf拷贝到这三个目录下，如图6-4所示。

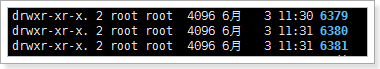


图6-5 Redis主从集群测试目录

进入这三个目录，分别修改配置文件，将端口分别设置为： 6379（Master）、6380（Slave）、6381（Slave）。同时要设置pidfile文件为不同的路径，如图6-5所示。

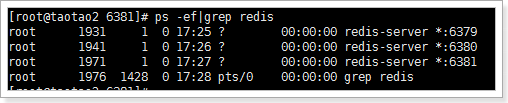


图6-6 端口配置图

分别启动三个redis实例，并在redis.conf中设置slaveof：slaveof <masterip> <masterport>。

2、系统测试方案

系统部署后主要测试系统前后端连通性、后端模块间消息发送性能、Jmeter压力测试。

（1）前后端连通性的测试方案及结果如下：

使用chrome浏览器的postman报文发送插件，选取系统的用户帐户查询接口作为本次测试的目标web接口。

系统具体测试流程：首先是浏览器端向服务端前置接口发起http协议的post请求，服务端前置层接收到http请求后，将消息中相应参数转化为dubbo请求，并向后端相应模块发送帐户查询请求，后端处理完将结果返回服务端前置，服务端前置再将结果返回给浏览器端。其次是在postman中发送post请求，并得到相应的json返回值，即表示功能性测试通过。最后编写post请求参数和填写对应的web接口url地址。本次测试的参数如表6-1所示。

表6-1 使用http协议post请求测试web接口

|  |  |  |
| --- | --- | --- |
| 接口地址 | Web接口url地址 | https://www.telemarket.com/v1/account |
| Post请求参数  form-data | customerId | 13201029118 |
| accessNumber | 15312667280 |
| method | queryAccount |

Postman测试请求发送截图如图6-7所示：

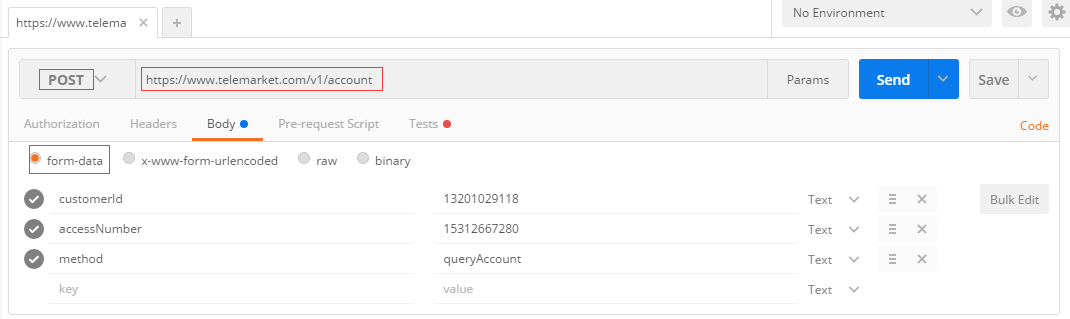


图6-7 postman发送http协议post请求

测试结果：成功获取返回JSON格式的response。平均响应时长为109ms。

（2）后端模块间消息通信性能测试方案及结果如下：

后端各模块间使用dubbo作为SOA服务治理框架，互相之间进行同步通信。本次采用订单管理模块向客户管理模块发起dubbo请求作为测试样例。测试主要流程为，调用订单管理模块的客户信息查询功能，订单模块会调用dubbo接口向客户管理模块发起查询，若返回正确结果则调用成功。

测试时，首先启动客户管理模块和Zookeeper注册中心，配置好Dubbo环境。

在订单管理模块中新建JUnit单元测试类，在测试类中新建客户信息查询方法queryCustomerInfo()，引入客户接口代理类CifCusProdFacadeServiceProxy,传入customerID参数，调用客户接口代理类CifCusProdFacadeServiceProxy的queryCustomerInfo方法。

测试结果表明，单元测试能够成功返回用户信息，模块间能够稳定地以Dubbo接口的形式进行通信。

（3）Jmeter压力测试方案及结果如下：

使用Jmeter进行多线程模拟并发，用以测试系统上线之后是否能够满足线上大并发量的要求。

启动Jmeter应用，创建线程组模拟用户并发访问，启动http请求采样，添加监听报告并点击启动测试。Jmeter应用界面如图6-8所示。

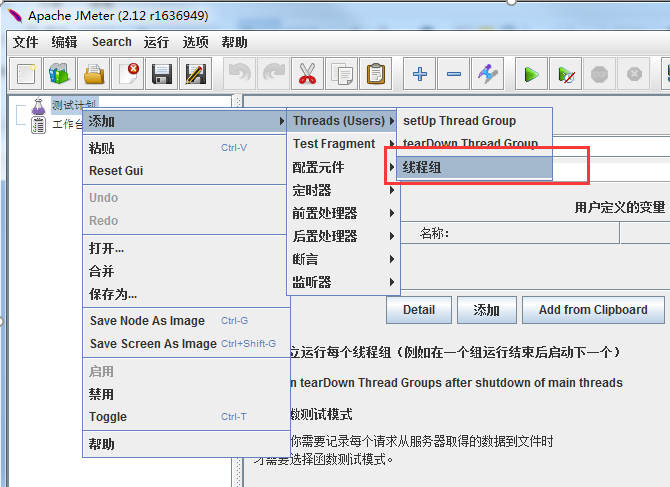


图6-8 Jmeter应用窗口

使用聚合报告和表格查看测试结果如图6-9所示。

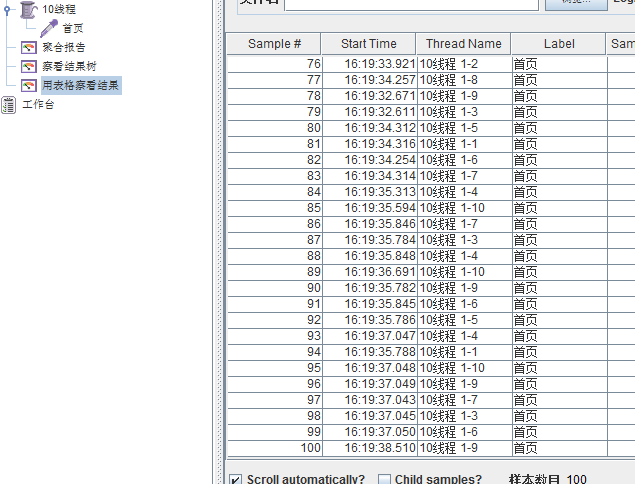


图6-9 Jmeter压力测试结果

压力测试结果表明：在高并发场景下，系统的平均响应时长能达到200ms。与传统的垂直架构应用相比较，在大并发量的场景下，采用SOA服务治理方式开发的本系统有更好的性能表现。

## 6.4 本章小结

本章对项目进行了系统化部署。主要部署了分布式生产环境，包括RabbitMQ的搭建、Redis主从集群的搭建、Nginx反向代理服务器的搭建，综合上一章所阐述的Dubbo环境搭建和部署，至此能够完整地搭建好项目的实际运行环境。

在项目环境搭建完毕后，需要进行实际运行性能测试。本章针对项目中主要的两个功能性接口进行了能力测试，对全局系统进行了模拟并发压力测试。在模拟压力测试的结果中，可以得出结论：采用分布式SOA服务治理的模式开发的精扫系统能够有效地应对大并发、高流量地客户访问。在实际的应用场景下，也能很好的应对生产环境所带来的压力，有效提升电信级大规模应用的性能。

# 第七章 总结与展望

7.1 论文工作总结

本文主要工作，提出了采用主流的分布式开源服务框架，构建了电信级客户精扫系统。系统开发过程中结合了创新型的前中后三层分层模型，底层以Dubbo框架为依托，以RabbitMQ为异步框架、Zookeeper为服务注册中心，构建了一个完整的、具有SOA服务治理特点的分布式系统。主要完成以下工作：

1、利用大量文献资料，研究了最新的主流分布式框架的发展和体系结构，阐述了使用分布式框架开发企业级应用的价值和相关实现技术。

2、针对目前最主流的开源分布式框架技术进行研究和原理讲解。

3、对目前构建电信级精确营销系统的应用需求、课题的实践来源、实施可行性等方面进行了阐明。对用例模型、数据模型在开发前期进行了详细的设计规划。

4、从分布式构建系统的特点出发，从系统整体规划上采用清晰的分层结构，针对系统开发过程中集成分布式框架的问题引入Dubbo这一SOA服务治理框架，并与Zookeeper、RabbitMQ等相关分布式技术进行整合，并对系统开发的具体流程进行了详细的阐述。

采用分布式服务框架、并具有SOA服务治理特征的应用开发是企业级项目开发的主要演进模式，本文在对客户精扫系统的构建探索过程中，也逐步总结出了完善的分层模型和项目架构设计经验，对项目前期的需求采集分析和技术基础积累、项目进行中的具体编码过程都经历了完整的实践。

7.2 展望

目前基于分布式架构的客户精扫系统，初步实现了前后端分离，移动端独立开发、后台端也独立开发与部署，初步完成了客户信息管理、订单管理、市场营销、数据报表等相关模块的功能。但是，在前端层的具体编码开发上，还有待开发实现。在前端的订单下单、产品展示、客户信息管理、意向收集等方面需要进行业务解耦，在与服务端前置的交互过程、http消息通信需要针对性设计，对于前端的技术选型和架构设计也将对未来客户体验和移动web应用的效率产生直接的影响。在后续研发过程中，如何完善移动web前端将是工作的重点。

另一方面，系统与运营商的传统CRM、计费账务等下游系统对接也是系统实施的关键，需要实现自动化、可调度、全流程管控等目标。

致 谢

能顺利完成这篇论文，首先要感谢我的导师蒋夏军老师的悉心指导与斧正。蒋老师渊博的知识和严谨的态度，帮助我开阔了学术思路，规范了工作方法，这些使我在工作和学习中受益匪浅！

同时，我要感谢南通电信和中国电信集团天翼电子商务公司上海技术部的同事们，他们在理论和技术上的通力支持与无私指导，帮助我在繁忙工作之余顺利完成论文。特别要感谢的还有周国均、吴超、路龙飞、朱伟炜、袁向飞等专家和同事。

此外也要感谢江苏电信公司、中国电信集团翼支付公司以及上海电信学院的一些专家的帮助与指导。

还要感谢参考文献的作者们，他们的学术见解和理论研究使我深受启发。

参 考 文 献

1. [中华人民共和国工业和信息化部办公厅](http://www.dangdang.com/author/%D6%D0%BB%AA%C8%CB%C3%F1%B9%B2%BA%CD%B9%FA%B9%A4%D2%B5%BA%CD%D0%C5%CF%A2%BB%AF%B2%BF%B0%EC%B9%AB%CC%FC_1). 中国电信业发展指导[M]. [人民邮电出版社](http://www.dangdang.com/publish/%C8%CB%C3%F1%D3%CA%B5%E7%B3%F6%B0%E6%C9%E7_1). 2014.5
2. [全国信息技术标准化技术委员会SOA标准工作组](http://www.dangdang.com/author/%C8%AB%B9%FA%D0%C5%CF%A2%BC%BC%CA%F5%B1%EA%D7%BC%BB%AF%BC%BC%CA%F5%CE%AF%D4%B1%BB%E1SOA%B1%EA%D7%BC%B9%A4%D7%F7%D7%E9_1). [中国电子技术标准化研究所](http://www.dangdang.com/author/%D6%D0%B9%FA%B5%E7%D7%D3%BC%BC%CA%F5%B1%EA%D7%BC%BB%AF%D1%D0%BE%BF%CB%F9_1). 中国SOA最佳应用及云计算融合实践[M]. [电子工业出版社](http://www.dangdang.com/publish/%B5%E7%D7%D3%B9%A4%D2%B5%B3%F6%B0%E6%C9%E7_1). 2012年03月
3. 吴洪. 国外电信管理[M]. 北京: 北京邮电大学出版社，2012
4. [范波勇](http://www.dangdang.com/author/%B7%B6%B2%A8%D3%C2_1). 电信营销与渠道建设[M]. [人民邮电出版社](http://www.dangdang.com/publish/%C8%CB%C3%F1%D3%CA%B5%E7%B3%F6%B0%E6%C9%E7_1). 2014.2
5. 郦军. 网上营业厅的发展思路[J]. 通信企业管理. 2011，(4):90-92
6. 肖金学. 电信企业营销管理[M]. 北京: 北京邮电大学出版社. 2011, 23-25
7. 张国强, 张晓诺. 网上营业厅的设计与实现[J]. 电信工程. 2011, (20):45-47
8. 泰勒. 电信网上营业厅安全不容忽视[J]. 计算机安全. 2014, (2):89-92
9. Thomas Erl. SOA Principles of Service Design[M]. Prentice Hall. 2008.06
10. [Thomas](http://www.dangdang.com/author/Thomas_1) Erl. SOA设计模式[M]. [科学出版社](http://www.dangdang.com/publish/%BF%C6%D1%A7%B3%F6%B0%E6%C9%E7_1). 2012.3
11. 王玉娟. 基于SOA的科技管理BPM平台设计与实现[J]. 计算机科学. 2013.11(40):423-425
12. 沈惠璋, 赵继娣. 基于SOA的分布式服务供应链信息共享平台研究与实践[J]. 计算机应用研究2010.2(27):606-610
13. 杨恒宇. 基于SOA的Web应用系统的研究与实现[D]. 合肥工业大学. 2006.5
14. 熊俊. 基于XML/Web服务的动态电子商务系统的关键性问题研究[J]. 软件学报. 2006.7
15. 张潇元, 刘利人, 韩海雯. 基于SOA云架构的电子监察业务平台的设计与构建[J]. 计算机科学2014. 11(41): 473-477
16. 彭渊. 大规模分布式系统架构与设计实战[M]. 北京: 机械工业出版社 2014.2
17. 李银胜, 柴跃廷, 沈卫明等. 面向服务架构与应用[M]. 北京: 清华大学出版社. 2008.9
18. 顾宁, 刘家茂, 柴晓路. Web Services原理与研发实践[M]. 北京: 机械工业出版社. 2006．
19. 于立群. 某电信运营商CRM销售管理子系统的设计与实现[D]. 北京邮电大学. 2012.5
20. 范强. 中国电信集团级CRM项目移动网厅模块的设计与实现[D]. 西安电子科技大学. 2011.6
21. 杜剑勐. 分布式架构在电信CRM系统中的研究与应用[J]. 中国高新技术企业. 2014(27):61-63
22. 陈康贤. 大型分布式网站架构设计与实践[M]. 北京: 电子工业出版社. 2014.9
23. George Coulouris,Jean Dollimore, Tim Kindberg. 分布式系统概念与设计[M] 北京. 机械工业出版社. 2004.1
24. 曾宪杰. 大型网站系统与Java中间件实践[M]. 北京: 电子工业出版社. 2014.4
25. 高哲. 基于SOA软件架构的研究与应用[D]. 武汉理工大学, 2007.5
26. 王玉娟. 基于SOA的科技管理BPM平台设计与实现. 计算机科学. 2013.11:423-425
27. 夏爱民，程芳，张宏志. 基于SOA的应急指挥信息服务系统设计与应用[J]. 通信技术. 2014.12: 1419-1423
28. 覃冠日, 朱晓民. 一种网管系统配置管理的设计方案[J]. 电信工程技术与标准化. 2016.08:62-66
29. 尹培培. 云计算与SOA在信息化建设中的融合应用[J]. 2014.3:26-29
30. 闵媫. 电信系统CRM中三户模型的设计与实现[D]. 山东大学. 2015.11
31. 冯博. 基于PaaS的电信虚拟运营商系统设计和实现[D]. 中国科学院大学. 2015.10
32. Philippe Kruchten. Casting Software Design in the Function-Behavior-Structure Framework[J]. IEEE Software. 2005,22(4):78-83
33. Ion LUNGU. Nicolae MERCIOIU.Database Access through Java Technologies[J]. Database Systems Journal 2010.1(1):9-17
34. 石光. 基子Web的电信计费系统管理软件的设计与实现[D]. 北京交通大学. 2015.6
35. 郝俊. 基于SOA三层模型的系统架构及建模平台设计[D]. 太原理工大学. 2015.5
36. Wei P, Hai-shan C, Zhao C. Research and Design of Web Application SystemDevelopment Platform[J]. Computer and Modernization, 2009.3
37. Yildirim U, Campean I F. Development of a structured approach for decomposition of complex systems on a functional basis[J]. IOP Conference Series: Materials Science and Engineering, 2014, 65(1):381-392.
38. 苑兴燕. UML 业务流程建模及其基于 Eclipse 开发工具的集成研究实现[D]. 北京邮电大学. 2013.5
39. Chun-shui X, Zhong-yu C, Ting G. Research on Component-based Workflow Modeling Method[J]. Computer and Modernization, 2013.
40. H F L, Mohamed H, El-Amine H M. A Generic Model for SOA Governance[J]. Mechatronics Engineering, Computing and Information Technology, 2014.
41. 田靖. 基于SOA的企业应用集成研究与实现[D]. 武汉理工大学. 2007.5
42. 卫程. 基于SOA的人力资源管理系统设巧与实现[D]. 山东大学. 2015.4
43. 魏钰媫. 基于SOA的人力资源信息管理系统设计与实现[D]. 西安电子科技大学. 2015.4
44. Huang Y, Kumaran S, Chung J. A service management framework for service-oriented enterprises. Proceedings of the IEEE International Conference on E-CommerceTechnology. Beijing, China. 2004:181-186.
45. Dang G, Cheng Z, Jin S et al. A service-oriented architecture for tele-immersion. Proceedings of the 2005 IEEE International Conference on e-Technology. e-Commerce and e-Service. Hong Kong, China. 2005:646-649.
46. 周彬. 基于SOA架构的高并发电子商务平台的构建[D]. 北京工商大学. 2016.1
47. Judith Hurwitz, Robin Bloor, Marcia Kaufman, Fern Halper. Service Oriented Architecture(SOA) For Dummies[M]. For Dummies. 2013．
48. Hugh Taylor, Angela Yochem, Les Phillips, Frank Martinez. Event-Driven Architecture: How SOA Enables the Real-Time Enterprise[M]. Addison-Wesley Professional. 2013．
49. George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair. Distributed Systems: Concepts and Design[M]. Addison-Wesley. 2013．
50. Dieter Fensel. Enabling Semantic Web Services[M]. Springer. 2010.
51. 韩丁, 沈建京，万芳. 基于SOA的服务构件封装技术研究[J]. 计算机工程与设计. 2009,30(7):1756-1759.
52. 马明. 关于Web应用架构Web Services的探究[J]. 电子商务. 2011,(10):51-52．
53. 凌晓东. SOA综述[J]. 计算机应用与软件, 2007,24(10):122-124．
54. 徐建. 基于SOA架构的移动网上营业厅系统研究与开发[D]. 电子科技大学. 2015.4
55. 詹鹏飞. 基于Web即时通讯的协作平台设计[J]. 数字技术与应用, 2012,(5):78-81
56. Bregon A, Daigle M, Roychoudhury I, et al. An event-based distributed diagnosis framework using structural model decomposition[J]. Artificial Intelligence, 2014,210(3):1–35.
57. 苑兴燕. UML业务流程建模及其基于Eclipse开发工具的集成研究实现[D]. 北京邮电大学. 2013.5
58. H F L, Mohamed H, El-Amine H M. A Generic Model for SOA Governance[J]. Mechatronics Engineering, Computing and Information Technology, 2014.